The Pennsylvania State University

The Graduate School

College of Engineering

ONLINE CUSTOMER REVIEW ANALYTICS

A Thesis in

Industrial Engineering

by

Visanu Chumongkhon

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

August 2019

The thesis of Visanu Chumongkhon was reviewed and approved* by the following:

Soundar R.T. Kumara
Allen E. Pearce/Allen M. Pearce Professor of Industrial Engineering
Thesis Advisor

Kamesh Madduri
Associate Professor of Computer Science and Engineering

Ling Rothrock
Interim Department Head and Professor of Industrial and Manufacturing Engineering

*Signatures are on file in the Graduate School

ABSTRACT

Analysis of customers reviews has become a valuable asset for business owners due to the significant increase in the volume of user reviews in online shopping platforms and marketing influencers' blogs. Automated text analysis plays an important role in measuring customers' satisfaction as well as identifying the topic of the complaints in real time. This study proposes ways to design and select algorithms for text analysis and aims to solve two problems: 1) sentiment analysis, and 2) topic labeling. The study is conducted on online reviews datasets. The 5-class sentiment analysis was experimented on the Amazon Fine Food Reviews dataset, whereas the 11-class topic labeling was analyzed on US Consumer Finance Complaints dataset. This research applied several of natural language processing techniques in text preprocessing together with conventional machine learning (i.e. SVM, Naïve Bayes, KNN and Logistic Regression) and deep learning models (i.e. LSTM and 1D CNN) to achieve the goal in sentiment analysis and topic labeling. In addition, one popular rule-based model sentiment analysis (i.e. VADER) is also performed. Result showed (1) the performance of VADER was relatively poor when compared to machine learning models in 5-classes sentiment classification; (2) SVM with bi-gram vectorizer achieved the best accuracy and macro F-1 score among all models in both sentiment classification and topic labeling problems, closely followed by the LSTM and 1D CNN with 100-dimension word embedding vectors; (3) SVM tends to learn better than other models for small classes; (4) one-versus-rest strategy on conventional models did not show any significant improvement in the case of extremely imbalanced dataset.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor Dr. Soundar Kumara, Professor of Industrial Engineering, for his valuable support and exemplary guidance throughout the course of my thesis. I am also grateful to him as his teaching has helped me acquire great knowledge and skills.

I would also like to express my gratitude to Dr. Kamesh Madduri, Associate Professor of Computer Science and Engineering, for taking his time to review my thesis and provide constructive feedback.

I take this opportunity to thank my colleague Rutvik Sharedalal for sharing his knowledge and giving some insightful inputs as well as my friend Sen Lin for the guidance in the programming.

Finally, I express my profound gratitude to my family and friends for their continuous support and encouragement throughout my graduate studies. This would not have been possible without all of you.

Chapter 1
Introduction

Text analytics is now becoming more of an asset to business analytics due to the increasing volume of customer reviews on social media and online platforms. The insights gained from this information have driven businesses into a new domain of customer understanding. It gives the insights of customer understanding, e.g. measuring the sentiment of a customer, measuring the helpfulness and subjectivity of the comments, and classifying the topics of what customers are saying about. A study of influencer marketing on Instagram indicates that consumers are more likely to trust suggestions from third-party bloggers or influencers than traditional advertising (Biaudet S., 2017). Moreover, a study conducted by Salehan et al. (2014) has shown that customers show more trust towards the online consumer reviews than the information provided by the vendors.

Sentiment analysis helps business firms in taking decisions by providing a better understanding of customer opinions for a product as well as competitor's products (Ghag et al., 2015). It can help the companies that receive feedback from social media such as Twitter, Instagram to quickly detect unhappy clients and not leave them unanswered for too long.

On the other hand, identifying what is a customer's problem is crucial in helping business firms in making decisions on solving customer's problem considering the importance, urgency and how much each problem could affect their business. Topic labeling is a popular approach to solve the problem by uncovering latent topics from the online reviews.

1.1 Motivation

Automated text analysis has become a key component in managing information for business firms due to the rapidly growing use of World Wide Web and digital devices. The study by Chevalier et al., (2006) on online review from Amazon.com and Barnesandnoble.com has shown that the relative sales of a book across the two sites are related to the differences across the sites in the average star ranking and the number of the reviews. Amazon.com has more lengthy and more positive reviews on an average and it has a higher total number of sales than its counterparts. This highlights the importance of customer review features on sales or revenues of online reviews, especially the customer's sentiment towards the products.

On the other hand, topic categorization can help in enhancing customer experience by automatically classifying the topic of complaints and assign support tickets to the right experts to resolve the issue in lesser time compared to manually assigning the ticket. Research from Hubspot company shows that 93% were more likely to be repeat customers at companies with excellent services, furthermore, 80% of the respondents said they had stopped doing business with poor customer experience (http://blog.hubspot.com)

The importance of building an automated text-based classification system has led to a body of research focused on sentiment analysis and topic labeling which are crucial tasks in text classification of online reviews.

1.2 Problem definition

The goal of this thesis is to develop automated text-based classification systems for two different tasks, one is to accurately predict the sentiment of Amazon online consumer reviews, the other one is to correctly classify the topic of online US Consumer Finance complaints.

For both the problems considered, the idea is to perform multi-class classification using the combination of text-based features and machine learning classification algorithms. For the Amazon online reviews, the classes range from '1 star' to '5 star' which refers to the range of customer sentiment 'Very Bad' to 'Very Good'. For the US Consumer Finance complaints, there are 11 target classes which refer to 11 different topics of the complaints from the customers.

Multiple types of text-based features will be used for this study which includes matrix based vectorized features, word embedding based features, semantic features. The algorithms used for classification purpose are: K-Nearest Neighbors, Multinomial naïve Bayes, Support Vector Machine, Logistic regression, 1D Convolutional Neural Network, Long short-term memory neural network and VADER. This thesis also discusses the required text pre-processing methods as they help avoid overfitting and reduce computational complexity by removing noise from the text such as common words with no or less contribution.

1.3 Organization of the thesis

The rest of the thesis is structured as follows. Chapter 2 discusses the previous work and provides an explanation of the features and algorithms. Chapter 3 provides an overview of the data. The methodology is discussed in detail in chapter 4. Chapter 5 includes a detailed description of the analysis and summarizes the results. Conclusions of the study and the scope of potential future work are discussed in chapter 6.

Chapter 2
Literature survey and Background

2.1 Literature survey

2.1.1 Research in sentiment analysis

The study conducted by Hutto et al. (2014) has shown that Valence Aware Dictionary and sEntiment Reasoner (VADER) shows high accuracy in classifying the sentiment of social media data. This parsimonious rule-based model incorporates a "gold-standard" sentiment lexicon that is tuned to micro-blog like contexts. In the 3-class sentiment classification problem on 4,000 tweets from Twitter social media, VADER model outperforms individual human raters with an F1 Classification Accuracy of 0.96 and 0.84 respectively. It also outperforms 11 other respectable sentiment analysis techniques, including Naïve Bayes and Support Vector Machine. Moreover, it is fast enough to be utilized online with real-time streaming data.

Ghag et al. (2015) showed that removing the stop words can increase the accuracy for traditional sentiment classifier models. This study was conducted on movie review dataset which is class-balanced by having 1,000 negatives and 1,000 positive data points. The traditional model, delta TFIDF, shows the increased accuracy from unprocessed dataset to stopwords removed dataset for traditional models as opposed to the other two sophisticated models proposed by the authors that did not yield a significant increase in accuracy. Therefore, stopwords removal can be treated as same as a hyperparameter.

Sentiment analysis on Amazon Fine Food Review conducted by Wu and Ji (2016) revealed that it is hard to distinguish reviews with close scores such as a review score 4 vs 5 and review score

1 vs 2. Therefore, in their study, the response classes are binarized to be either positive or negative class by aggregating reviews score 4 with score 5 into the positive class, and grouping reviews score 1 with score 1 into the negative class, and ignore samples with score 3. Further, they proposed Recursive Neural Network for Multiple Sentences (RNNMS) that handles multiple sentences at once, as opposed to most RNNs that are designed to only consider one single sentence as input. The proposed model applied on 50-dimension GloVe word vectors trained by twitter data outperformed the Naïve Bayes classifier using averaged word vectors.

The focus of previous research on sentiment analysis was largely centered around on automatizing the prediction of review helpfulness to tackle the issues regarding the enormous quantity of online reviews.

2.1.2 Research in text classification

Multi-Channel Convolutional Neural Network in text classification was first proposed by Kim, Y., (2014). In his study, two channels of input to CNN are cosists of (1) static channel which the weights of word enbedding vectors remain the same (2) non-static channel which the the word embeddings will be adjusted during the training process. The result from both channels, are concatenated into a single vector at the flatten layer. He also compared the accuracy of the above method with one-channel CNN models which are (1) static word embedding CNN; (2) non-static CNN; and (3) randomly-initialized. Filter sizes of 3,4,5 are used in each channel. Google News word2vec, which is trained from 100 billion words, was used as the word embedding in his static and non-static CNNs. The concept of multi-channel CNN from Kim, Y., (2014) made use of both pre-trained and task-specific vectors

Zhang & Wallace, (2016) did an extensive analysis based on the idea from Kim, Y., (2014). The results showed that the filter region sizes and number of feature maps can make significant impact on performance, whereas regularization has relatively small impact. Moreover, 1-max pooling performed better than other max pooling strategies.

Various text-based features and machine learning algorithms used in the domain of spam review detection are discussed here. This section also emphasizes the use of Support Vector Machine (SVM) in text categorization problems.

Crawford et al., (2015) conducted a survey of spam detection in which they discussed features extracted from text using Natural Language Processing and compared different approaches for classifying the reviews as ‘spam’ or ‘not spam’. Features used in the study included Bag of words, Term Frequency, POS tagging, and other lexical and semantic features. The classification task was performed using algorithms such as SVM , Naïve Bayes, and Logistic Regression. Among the algorithms used, SVM was found to generally outperform Naïve Bayes and Logistic Regression while occasionally beaten by them. Moreover, the performance was shown to be increased when multiple features were combined.

Shojaee et al., (2013) conducted a research on detecting deceptive opinions. They came up with stylometric features to identify patterns in spammer writing style. It was stated that the spammers try to change their writing style and verbal usage by using simpler, shorter and fewer average syllables per word. The features were categorized into 2 types, i.e. lexical and syntactic features.

Example of some of these features include the total number of tokens, average sentence length, average token length, occurrences of uppercase, etc. The machine learning classification algorithms used were Support Vector Machine and Naïve Bayes. Features were analyzed separately as well as combined and compared using F-score measure. The highest accuracy was achieved when the features were combined irrespective of the algorithm applied. Moreover, SVM performed better overall compared to Naïve Bayes in all combinations of the features.

Previous research has also shown that combining the semantic features and word frequency-based features can outperform either of them used alone. In their study of text classification, Lilleberg et al., (2015) combined the word vectors obtained from the Word2vec algorithm with the TF-IDF measure of the words. The assumption was that the TF-IDF matrix does not represent the semantic relationship between the tokens which Word2vec provides. Besides this, various combinations were tried in terms of the inclusion of stop words. The '20 newsgroup' text data was used for the study and Linear SVC was used as the classification algorithm. Combining Word2vec with TF-IDF without the stop words rendered the highest accuracy among all the different combinations except in some rare cases where it could not outperform the TF-IDF without the stop words.

As far as machine learning classification algorithms are concerned, Naïve Bayes and Support Vector Machine perform better for the text classification problems (Vinodhini et al., 2012). Joachims (1998) has given theoretical evidence on why SVM performs well for text categorization. He tried to strengthen his argument by providing the following justifications:

1) SVM has the potential to handle large feature spaces related to the text categorization problems as its functionality does not depend upon the dimensionality of the problem.

2) SVM is well suited for problems with sparse instances and dense concepts (High dimensionality with few non-zero entries) such as text categorization.
3) The tendency of most of the text classification problems is to be linearly separable and the idea behind the SVM is to find such linear separators.

2.2 Vectorized features

One way to make the numerical representation of documents to work with classification algorithms is to use vectorization process. This method is also known as "Bag of Words". We can construct a weight matrix that contains the values of relations between each unique words and documents. Each vector is represented as a vector in n-dimensional vector space. We can imagine a matrix A with a dimension of MxN as shown in Figure 2-1, where N is the number of unique words in the corpus of all documents. M is the number of documents. Each row represents a document vector. Each element a_{ij} represents weight value of word j in document i. Next, we will discuss variants of the representation.

a(0,0)	a(0,1)	a(0,2)	…	a(0,j)	…	a(0,N)
a(1,0)	⋱					
a(2,0)		⋱				
⋮			⋱			
a(i,0)				a(i,j)		
⋮					⋱	
a(M,0)						a(M,N)

Figure 2-1: Weight matrix for vectorized features (Bag of Words) method

The weight matrix is used as an input to classification algorithms. The limitation of this feature is that it does not consider the semantic relation between the terms as it counts the frequency of each word separately. n-gram is a way to deal with the semantic relation, which we will discuss later in this section.

2.2.1 Term Binary vectorizer

This method checks whether a particular word appears in a particular document. It is the most simple and easy to implement (Trstenjak et al., 2014). The value a(i,j) is set to 1 if the word j appear in document i, otherwise the value is set to 0.

2.2.2 Count vectorizer

This method is also known as Term Frequency vectorizer. It counts the number of occurrences of the word j in document i. The value in the matrix a_{ij} is shown below:

$$a_{ij} = f_{ij} = \text{frequency of term } j \text{ in document } i$$

Some document classification algorithms normalize the count vector by dividing with the frequency of the most common term in the document as shown below (Trstenjak et al., 2014):

$$a_{ij} = f_{ij}/max_i\{f_{ij}\}$$

2.2.3 TF-IDF vectorizer

Term frequency-inverse document frequency (TF-IDF) is another approach to characterize the documents. It assigns the weight to each term frequency. The weight of a word is inversely proportional to how often it occurs across all documents. TF-IDF is especially useful when stop words have not been removed from the corpus. This is because TF-IDF tends to lower the importance of stop words which are frequently but do not convey much meaning such as ‘a’, ‘an’,

'the', 'is', 'are', 'on', 'at'. In application, the term frequency is usually normalized by the document length (Raschka & Mirjalili, 2014):

$$\text{normalized term frequency} = \frac{tf(t,d)}{n_d}$$

The formula for TF-IDF can be presented mathematically as below:

$$\text{TF-IDF} = tf_n(t,d) \cdot idf(t)$$

where $tf_n(t,d)$ be the normalized term frequency; n_d is the number of documents; $n_d(t)$ is the number of documents that contain the term t; and idf is the inverse document frequency, which can be calculated as follows:

$$idf(t) = \log\left(\frac{n_d}{n_d(t)}\right)$$

2.2.4 n-grams

The disadvantage for vectorized features method is that it does not consider the semantic meaning of the group of words. n-grams considers n adjacent words from the document. By implementing n-grams, the model will learn the semantic meaning of pairs (2-grams), triplets (3-grams) and a group of adjacent n words (n-grams). For example, considering the sentence "*I scream, you scream, we all scream for ice cream*.", the word "*ice cream*" can be captured by 2-grams detector and it has unique meaning different from single words "*ice*" and "*cream*", "*But I don't know many people that scream for "cream." And nobody screams for "ice," unless they're about to slip and fall on it. So you need a way for your word-vectors to keep "ice" and "cream"*

together." (Lane et al., 2019). Similarly, the 2-grams "not good" has a negative meaning as opposed to the word "good" with a positive meaning.

Introducing n-grams creates a lot more feature columns, which in turn, can lead to overfitting the model. The appropriate range of n-grams is required to prevent the overfitting problem.

2.4 Word embedding based features

One benefit of using dense representation is computational: majority of neural network toolkits do not perform well with very high-dimensional, sparse vectors (Yoav, G., 2017). In the vectorization method described in the previous section, the number of features is equal to the total number tokens (words) including n-grams. It makes the input matrix a very sparse matrix. The main function of Word Embedding method is to reduce the number of features. Word Embedding represents each word as a dense vector. The word embedding vectors of 2 words that have a similar meaning, semantic and syntactic similarity will have small "cosine similarity". We will discuss 2 variants namely, Word2Vec and Embedding Layer.

2.4.1 Word2vec

Word2vec uses the weights of shallow neural networks to represent the vector of each word in the corpus. There are two architectures used to train Word2vec namely Continuous Bag of Words (CBOW) and Skip-gram (Mikolov et al., 2013).

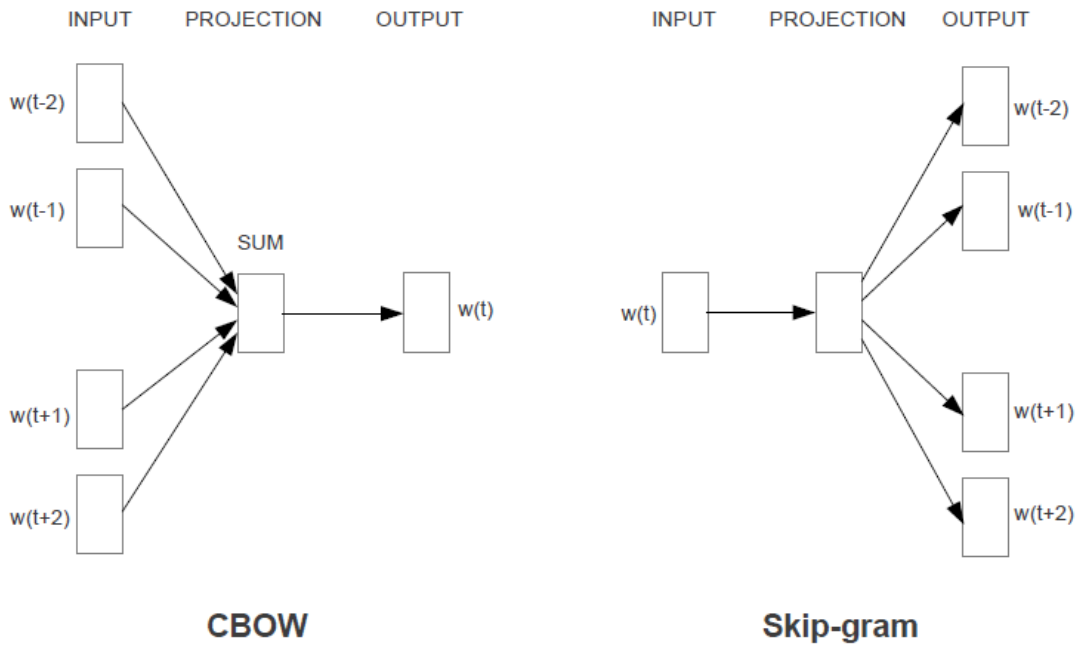


Figure 2-2: CBOW and Skip-gram model architectures (from Mikolov et al., 2013)

The idea of CBOW method is to train the Word2Vec model to predict a word based on the context words. Skip-gram method uses the current word to predict the context words as shown in Figure 2-2. After the training process, the weights between the projection layer and the output for the words which are supposed to be predicted will be taken as the elements of word vectors.

The input of the Word2vec model is the one-hot encoded of the words. Word vectors are trained using the back-propagation method and the Softmax function is used at the output layer to calculate the probability of the outcome (Rong X., 2014). Mikolov et al. (2017) described that Skip-gram is better when the analysis involves capturing the semantic relationship between the words. We can use the pre-trained Word2vec such as Global Vectors for Word Representation (GloVe) which is trained with the large dataset.

2.4.2 Embedding layer

Another way to make a dense representation of word embedding from our own training dataset is to use Embedding layers such as Keras API. This method creates a vector representation of each word which is fitted from our own corpus in the training dataset. The advantage of this method is that the word vectors are specifically attuned to our own topic and contexts from the training dataset. However, there is a drawback when the size of training dataset and vocabulary is not big enough. We need to consider those factors in choosing to use the pre-trained embedding layer or to train from our own corpus.

2.8 Machine learning algorithms for classification

2.8.1 K-Nearest Neighbors

K-Nearest Neighbors is a similarity-based learning based on the idea of nearest neighbor algorithm. The algorithm takes a weighted matrix which contains the weighted values (for example, TF-IDF) of the selected document and all other documents in the corpus. In the next step the algorithm will find the vector distances between the documents by using the following equation:

$$d(x,y) = \sqrt{\sum_{r=1}^{N}(a_{rx} - a_{ry})^2}$$

where $d(x,y)$ is the distance between two documents, N is the number of unique words in the corpus, a_{rx} is a weight of the term r in document x, a_{ry} is a weight of the term r in document y.

Next, It is necessary to determine K number which is the number of required number of documents from the corpus which is closest in distance to the selected document (Trstenjak et al., 2014). Defining the number K helps in mitigating against the impact of on individual noisy instances on the nearest neighbor algorithm (Kelleher, et al. 2015). Finally, the classification will return the majority of the target level within the set of K nearest neighbors.

$$\mathbb{M}_k(q) = \underset{l \in levels\ (t)}{\arg\max} \sum_{i=1}^{k} \delta(t_i, l)$$

Where $\mathbb{M}_k(q)$ is the prediction of the model $\mathbb{M}$ for the query q using k as the nearest neighbors parameter; levels(t) is the set of levels from target features; and l is an element of this set; t_i is the target for instance d_i; and $\delta(t_i, l)$ returns 1 if $t_i = l$, returns 0 otherwise.

2.8.2 Multinomial naïve Bayes

Naïve Bayes is a probabilistic classifier. It uses an assumption from Bayes' theorem that the occurrence of a current feature is independent of the occurrence of the other features. Multinomial Naïve Bayes classifier makes use of the term frequency rather than just using binary values (Raschka & Mirjalili, 2014). The individual likelihood can be calculated as below:

$$\widehat{\mathrm{P}}(x_i|c_j) = \frac{\sum tf(x_i, d \in c_j) + \alpha}{\sum N_{d \in c_j} + \alpha \cdot V}$$

The class-conditional probability of encountering the text **x** can be calculated as shown below:

$$\boldsymbol{P}(\mathbf{x}|c_j) = \boldsymbol{P}(x_1|c_j) \cdot \boldsymbol{P}(x_2|c_j) \cdot \ldots \cdot \boldsymbol{P}(x_n|c_j) = \prod_{\boldsymbol{t=1}}^{\boldsymbol{m}} \boldsymbol{P}(x_i|c_j)$$

where x_i is a word from feature vector **x** of a particular instance; c_j is a class of from the set $c_1 \ldots c_m$; $\sum tf(x_i, d \in c_j)$ is the sum of raw term frequencies of word x_i from all the documents belonging to class c_j; $\sum N_{d \in c_j}$ is the sum of all term frequencies in the training dataset of class c_j; α is an additive smoothing parameter ($\alpha = 1$ for Laplace smoothing); V is the size of vocabulary (total number of unique words in the training set). While calculating the IDF value, 1 is added in the denominator as a smoothing parameter to avoid division by zero in the case of an unseen word (Pedregosa et al., 2011).

From Bayes' theorem, the posterior probabilities can be simply calculated as the multiplication of class-conditional and prior probabilities divided by the evidence term. The evidence term is a constant. Hence, we can apply maximum likelihood to classify the documents (Raschka & Mirjalili, 2014).

2.8.3 Logistic regression

Logistics regression models the probability that the target belongs to a particular category (James et al., 2013). The basic logistic regression model is used in predicting binary dependent variable. To build a logistic regression model, we threshold the output of the linear regression model. Maximum likelihood method is used in fitting the model. Instead of the regression function being the dot product of the weights and the independent variables, such dot product is passed through the logistic function as described in the equation below (Kelleher, et al. 2015).

$$\boldsymbol{logistic}(\boldsymbol{x}) = \frac{\mathbf{1}}{\mathbf{1} + \boldsymbol{e}^{(-x)}}$$

For multiple logistic regression,

$$\mathbb{M}_w(\mathbf{d}) = \boldsymbol{logistic}(\boldsymbol{w} \cdot \boldsymbol{d})$$

$$= \frac{\mathbf{1}}{\mathbf{1} + \boldsymbol{e}^{(-\boldsymbol{w}\cdot\boldsymbol{d})}}$$

where **d** is a set of descriptive features for a sample, **w** is the set of weights in the model, $\mathbb{M}_w(\mathbf{d})$ refers to the output of a model $\mathbb{M}$ parameterized by parameters **w**. The predicted probabilities using logistic regression lies between 0 and 1 (James et al., 2013), Figure 2-3 shows a plot of the logistic function equation.

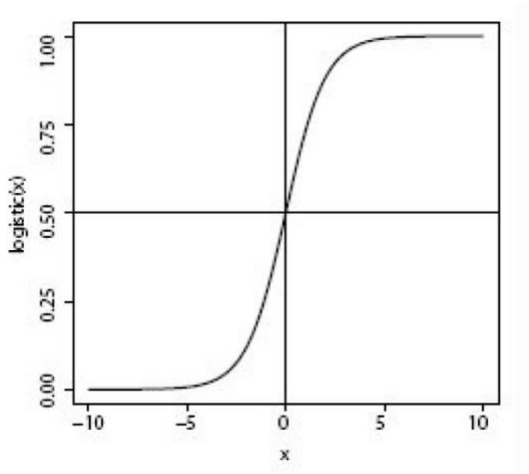


Figure 2-3: Logistic function (from Kelleher, et al. 2015)

For binary classification, the predicted probabilities are shown below:

$$\boldsymbol{P}(\boldsymbol{Class\ 0}|\boldsymbol{d}) = \mathbb{M}_w(\mathbf{d}) = \frac{\boldsymbol{e}^{(\boldsymbol{w}\cdot\boldsymbol{d})}}{\mathbf{1} + \boldsymbol{e}^{(\boldsymbol{w}\cdot\boldsymbol{d})}}$$

$$\boldsymbol{P}(\boldsymbol{Class\ 1}|\boldsymbol{d}) = 1 - \mathbb{M}_w(\mathbf{d}) = 1 - \frac{\boldsymbol{e}^{(\boldsymbol{w}\cdot\boldsymbol{d})}}{\mathbf{1} + \boldsymbol{e}^{(\boldsymbol{w}\cdot\boldsymbol{d})}}$$

Multinomial logistic regression, also known as Softmax regression, is a generalization of logistic regression to handle multi-class classification. The posterior probability of a class given a document is shown below:

Given a set of coefficients A and b, we assume the probability of observing label $\boldsymbol{y_i}$ is shown below (Aggarwal et al., 2012).

$$\boldsymbol{P(C = y_i|x_i) = \frac{e^{(A \cdot X_i + b)}}{1 + e^{(A \cdot X_i + b)}}}$$

Another good way to do multiple-class classification is to use "one-versus-rest" strategy (Kelleher, et al. 2015). In this approach, given that we have r target levels, we can create r models. Each of the models is one-versus-rest (binary) logistic regression model.

$$\mathbb{M}_{w_1}(\mathbf{d}) = \boldsymbol{logistic}(w_1 \cdot \boldsymbol{d})$$

$$\mathbb{M}_{w_2}(\mathbf{d}) = \boldsymbol{logistic}(w_1 \cdot \boldsymbol{d})$$

$$\vdots$$

$$\mathbb{M}_{w_r}(\mathbf{d}) = \boldsymbol{logistic}(w_1 \cdot \boldsymbol{d})$$

Basically, we make r models each of which is a binary logistic regression specifically attuned to predict each class. This method will be used in the text classification problem to compare the performance of one-versus-rest method against multinomial regression model. Not only logistic regression, but also KNN and Naïve Bayes model will be performed using this method to compare the accuracy in chapter 4 and 5.

2.8.4 Support vector machine

Support Vector Machine (SVM) is an extension of support vector classifier that results from enlarging the feature space in a specific way using kernels in order to deal with the non-linear boundary between the classes (James et al., 2013). SVM finds a separator in a feature space that can optimally separate different classes with maximum margin of separation.

The model first sets the negative and positive target feature level to -1 and +1 respectively. Then, it builds support vector machine prediction with negative target result in the model outputting ≤ -1. Similarly, it aims to have instances with the positive target values outputting ≥ +1. The gap between the output -1 and +1 is for the margin. The margin is the distance from the decision boundary to the nearest training sample (Kelleher, et al. 2015).

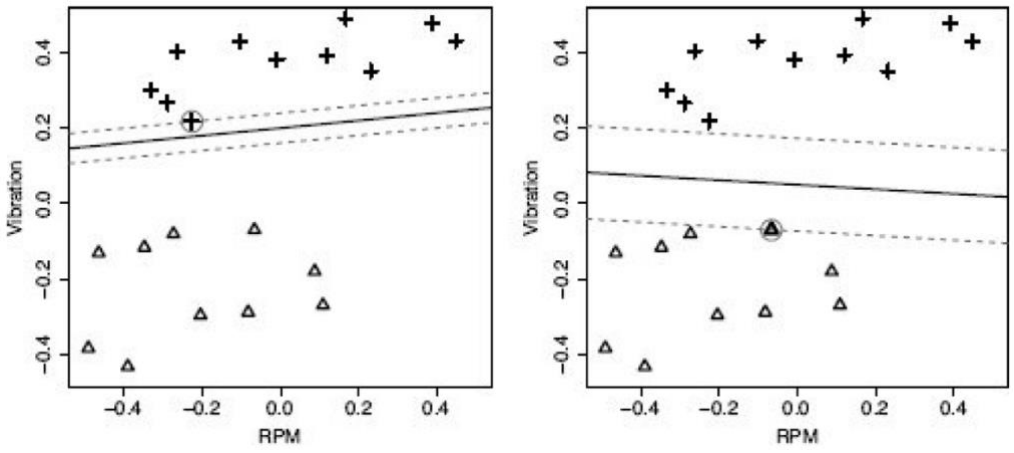


Figure 2-4: SVM decision boundaries (left) a decision boundary with small margin, (right) a decision boundary with a large margin (Kelleher, et al. 2015)

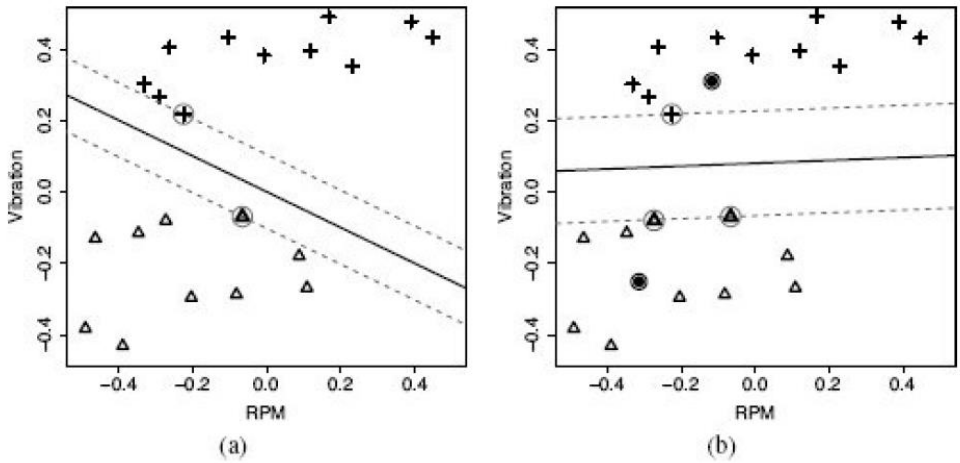


Figure 2-5: Trained SVM decision boundaries (left) a decision boundary with small margin, (right) a decision boundary with a large margin (Kelleher, et al. 2015)

The training process for SVM is done to find the best separating hyperplane using **constrained quadratic optimization. Figure 2-4** shows a decision boundary with small margin (left) and a decision boundary with a large margin (right) before the training process. **Figure 2-5** shows the optimized decision boundaries for both cases. The SVM is defined as

$$\mathbb{M}_{\alpha,w_0}(\mathbf{q}) = \sum_{\boldsymbol{i=1}}^{\boldsymbol{s}} (\boldsymbol{t_i} \times \boldsymbol{\alpha}[\boldsymbol{i}] \times (\mathbf{d_i} \cdot \boldsymbol{q}) + w_0)$$

where **q** is a set of descriptive features for a query instance; $(\mathbf{d_1}, \boldsymbol{t_1}), \ldots, (\mathbf{d_S}, \boldsymbol{t_S})$ are support vectors (instance composed of features $\boldsymbol{d_i}$ and a target feature $\boldsymbol{t_i}$). w_0 is the first weight of the decision boundary; and $\boldsymbol{\alpha}$ **refers to** a set of parameters determined during the training process (there is a parameter for each support vector α [1], …,α[s]); $\mathbb{M}_{\alpha,w_0}(\mathbf{q})$ refers to the output $\mathbb{M}$ of a model parameterized by parameters w_0 and α (Kelleher, et al. 2015). The constraints for the training process are

$$w_0 + w \cdot \mathbf{d} \leq -1 \text{ for } \boldsymbol{t_i} = -1$$

and

$$w_0 + w \cdot \mathbf{d} \geq +1 \text{ for } \boldsymbol{t_i} = +1$$

Talking about the objective function, since the perpendicular distance from any sample to the decision boundary is

$$\text{dist}(\mathbf{d}) = \frac{\text{abs}(w_0 + w \cdot \boldsymbol{d})}{||\boldsymbol{w}||}$$

where ||**w**|| is the Euclidean norm which can be calculated as

$$||\mathbf{w}|| = \sqrt{\mathbf{w}[1]^2 + \mathbf{w}[2]^2 + \cdots + \mathbf{w}[m]^2}$$

where **d** is a set of descriptive features of *m*; and t is a target feature.

The objective function is to maximize $\frac{\mathbf{2}}{||\mathbf{w}||}$ which is the boundary size of the margin as shown in **Figure 2-6**. It is equivalent to the objective function of minimize $\|\mathbf{w}\|$

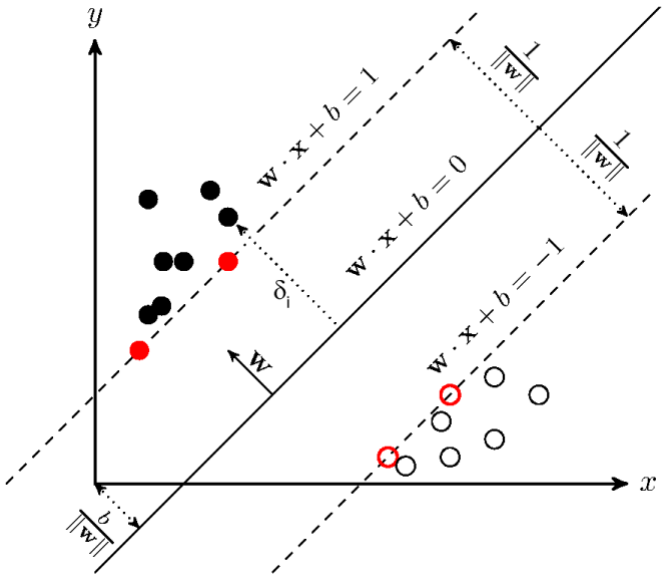


Figure 2-6: Maximum-margin hyperplane (line in 2-D) separating two classes data

The SVM with "kernel trick" utilizes kernel function to avoid a high computational effort from dot product operations of two high-dimensional vectors. The Linear kernel, Polynomial kernel and Gaussian radial basis kernel can be applied to the SVM as shown below:

$$\mathbb{M}_{\alpha,kernel,w_0}(\mathbf{q}) = \sum_{\boldsymbol{i=1}}^{\boldsymbol{s}} (\boldsymbol{t_i} \times \boldsymbol{\alpha}[\boldsymbol{i}] \times kernel(\mathbf{d_i} \cdot \mathbf{q}) + w_0)$$

Linear kernel: $kernel(\mathbf{d} \cdot \boldsymbol{q}) = \mathbf{d} \cdot \boldsymbol{q} + \mathrm{c}$; where c is an optional constant

Polynimial kernel: $kernel(\mathbf{d} \cdot \boldsymbol{q}) = (\mathbf{d} + \mathbf{q} + 1)^p$; where p is the polynomial degree

Gaussian radial basis kernel: $kernel(\mathbf{d} \cdot \boldsymbol{q}) = \boldsymbol{exp}(-)\mathbf{d} \cdot \boldsymbol{q} + \mathrm{c}$; where γ is a tuning parameter (Kelleher, et al. 2015).

Previous study has shown general agreement that linear kernel SVM works very well because of the high dimensionality of the feature set of the text data. On the contrary, the non-linear one does not pay for itself (Aggarwal et al., 2012).

2.8.5 Deep Learning Approaches

2.8.5.1 Convolutional Neural Network

Convolutional Neural Network (CNN) is a class of deep neural networks that is most commonly used in image analysis tasks such as object detection, segmentation, and image processing. Goodfellow et al. (2017) describes CNNs as neural networks that employ a mathematical operation called "convolution" in place of general matrix multiplication in at least one of their layers. In this thesis, we will discuss about three important concepts in CNN architecture: local receptive fields, shared weights and biases, and activation and pooling.

Antoine J. (2017) described that local receptive fields allow CNNs to recognize more and more complex patterns in a hierarchical way, by combining lower-level, elementary features into higher-level features. This property is called "compositionality". In a traditional neural network, neurons in the input layer are fully connected to neurons in the hidden layer. However, in CNN, only a small region of input layer neurons connect to neurons in the hidden layer. These regions are known as local receptive fields.

Second, CNN has weights and biase values that are same for all hidden neurons in a given layer. This characteristic can be called "local invariance" (Antoine, J. 2017). Local invariance and compositionality enable a CNN to detect a feature regardless of its position in the data matrix. For example, detecting edges or blob from an image, detecting some specific word pairs (bigram) that convey specific meaning from a document.

Third, Activation and pooling. The activation step applies the transformation to the output of each neuron by using activation functions (e.g. ReLu). CNN then further transforms the output by applying pooling. Pooling reduces the dimension of the feature map to a single output. Antoine J. (2017) suggests that global pooling works better than local pooling over the small region for short text classification. He also mentioned that maximum pooling works better than average pooling since we are interested in extracting the most salient feature from each feature map.

CNN can be constructed by a combination of the above three concepts. Moreover, convolutional layers may directly be stacked on top of each other as shown in Figure 2-7. Every hidden layer increases the complexity of the learned image features, for example, from learning how to detect the edges in the first layer to learning how to detect complex shapes in the last layer.

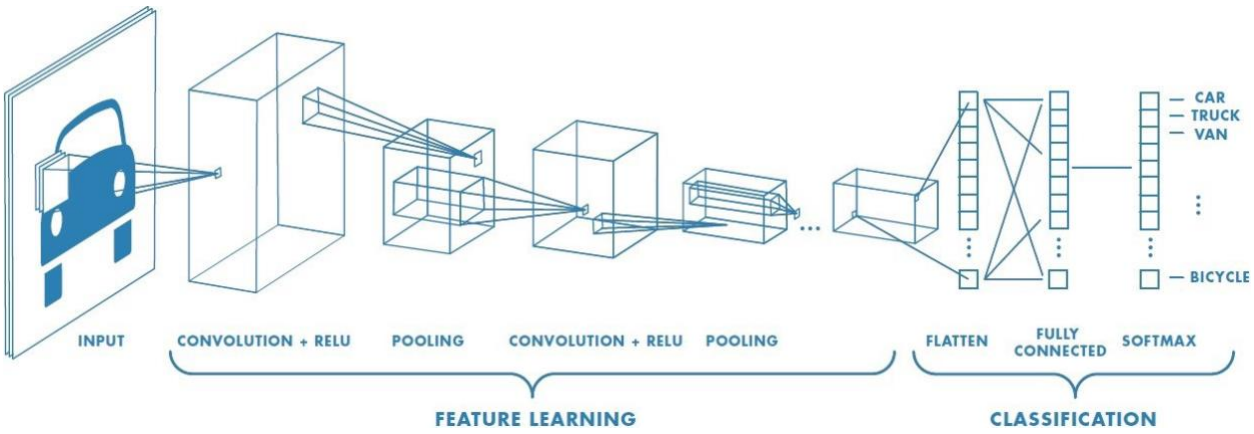


Figure 2-7: Example of 3-channel CNN for image classification (from http://mathworks.com)

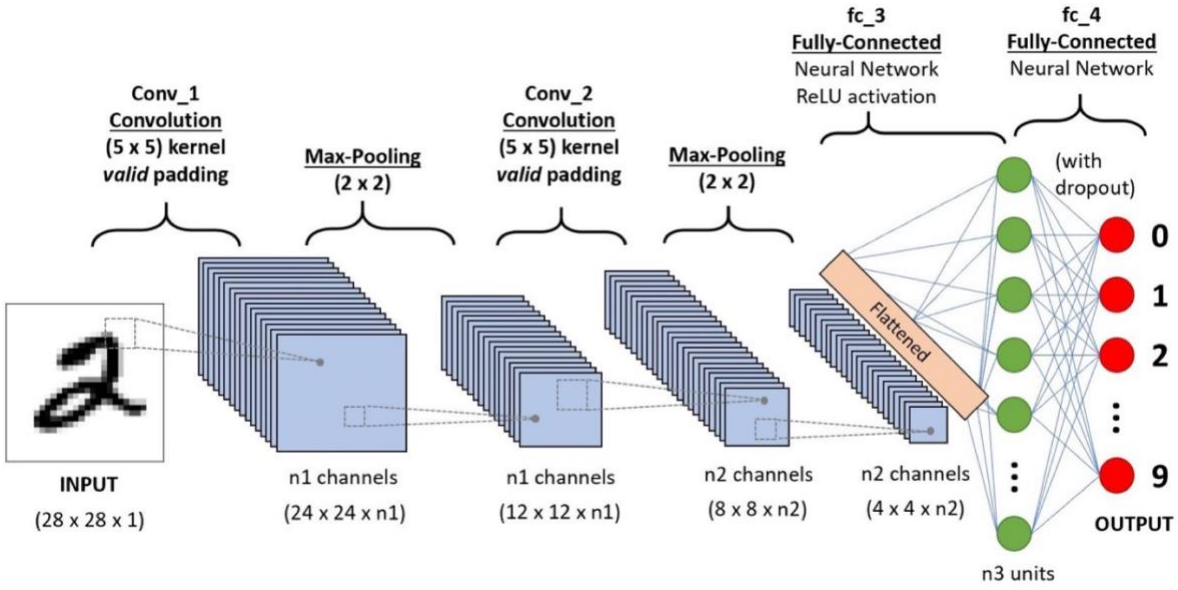


Figure 2-8: An architecture of CNN for 1-channel (black & white) image classification (from http://towardsdatascience.com)

Talking the case of text classification with CNN, all three concepts mentioned above can be applied. As deep learning methods prefer a more dense input, instead of sparse matrix like bag-of-words (e.g. count vectorizer, TF-IDF vectorizer), we can use pre-computed embedding such as Word2Vec (e.g. Stanford GloVe) or we can use Embedding Layer (e.g. Keras API) which is fitted from our own corpus in the training dataset.

After getting the word embedding, one thing we need to consider is padding. In text processing, the length of the reviews or documents is not always the same. Padding is added to shorten documents in order to transform all documents to have the same length. Also, padding is done in wide-format sentence convolution at the beginning and the end of the sentence as shown in Figure 2-9.

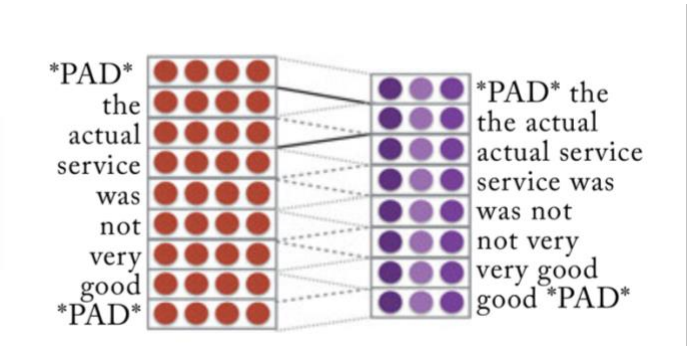


Figure 2-9: Sentence convolution with padding, window size (k) =2, dimensional output (*ι*) = 3 (from Yoav, G., 2017)

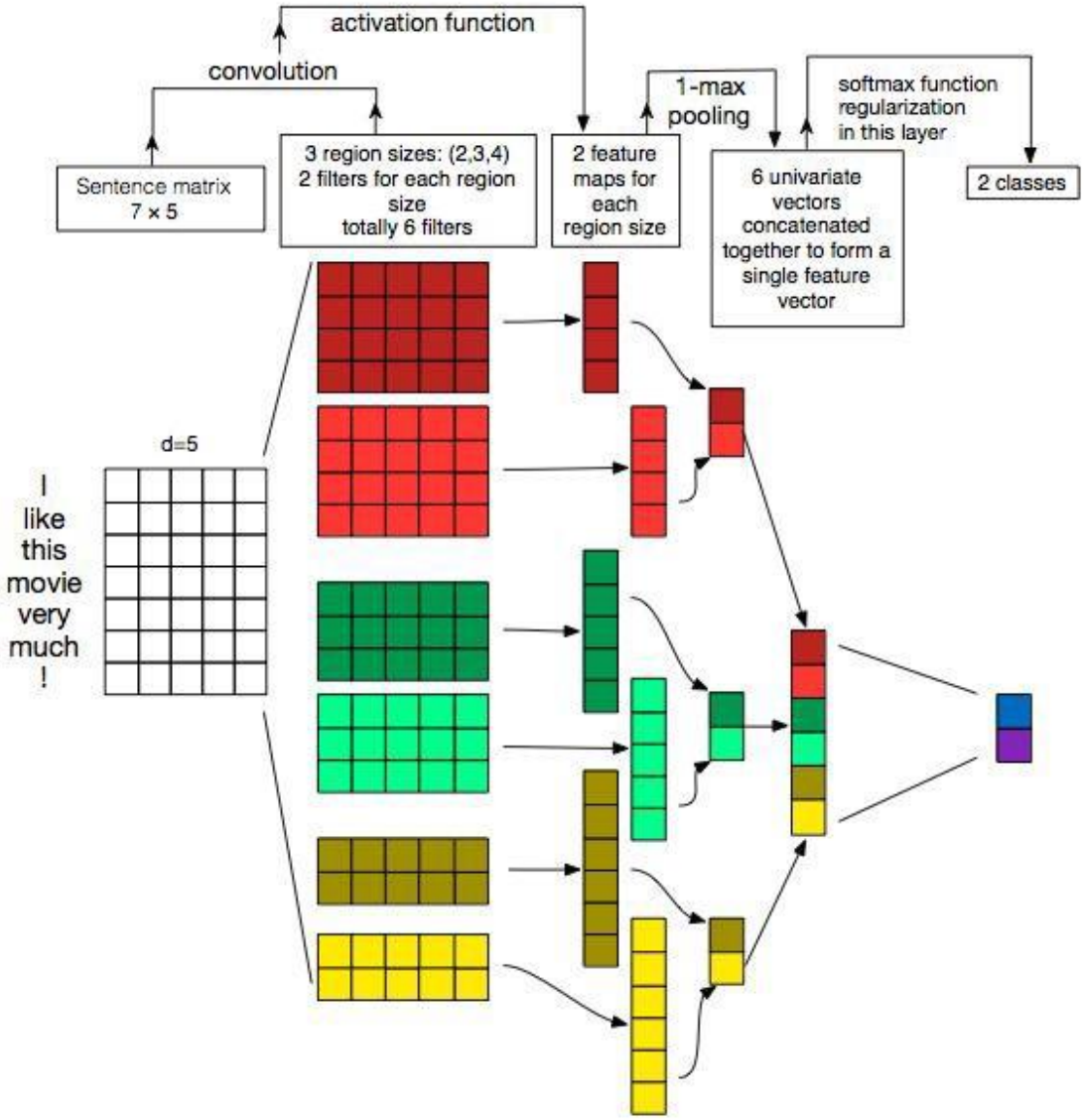


Figure 2-10: Architecture of CNN for text classification (from Zhang et al.,2016)

Figure 2-10 illustrates an example of the architecture of CNN for text classification having 1 channel input to the model (Zhang et al., 2016). In this example, convolutional filter size of 2,3 and 4 are applied to the input layer, each of which has 2 filters. The convolutional filter size of n can be interpreted as equivalent to n-grams in bag-of-words method, meaning they work similarly to bigram, trigram, and four-gram. By sliding a filter over the document length, the model calculates the weighted sum which is a dot product of the weights of the convolutional filter and the input. As we can see that in this figure, 2 filters are applied for each size. After calculated the dot products (weighted sum) for all filters and pass them through the activation functions, the model applies maximum pooling. It is basically choosing the highest activation for each filter. After convolution and max-pooling steps, the model passes the values the fully connected layer. Basically, in this step, it concatenates those highest activations from each filter together to further predict the result to the output layer. This example shows 2 classes classification using softmax function at the output layer. For binary classification like this example, it can also be designed as one node with sigmoid function.

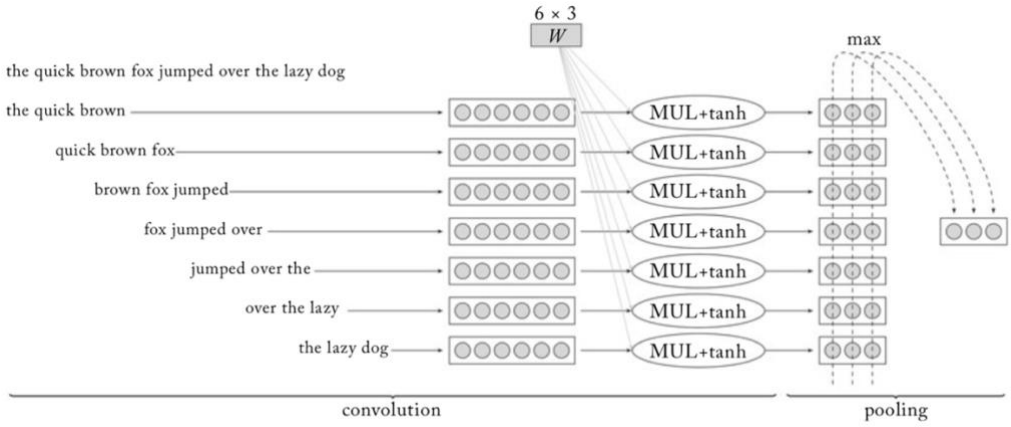


Figure 2-11: Maximum pooling (from Yoav, G., 2017)

Figure 2-11 explains the pooling for sentence convolution with window size k = 3. Each word is represented in 2-dim embedding vector. The embedding vectors are then concatenated, resulting in 6-dim window representations. Then, a 6x3 filter of linear transformation followed by element-wise tanh is applied to each of the 6- dim vector. Then, a max pooling is applied, taking the maximum value from each dimension, resulting in a 3-dim pooled vector (Yoav, G., 2017)

Talking about channels in image processing, as we can see in Figure 2-12, the input can be of 3 channels representing pixels of R, G and B color. In sentence classification problem, we can make multi-channel input by adding part-of-speech tag (POS tag) in addition to the first channel which is the sequence of words (Yoav, G., 2017). Moreove, shape of the words can be added as another channel to make the input matrix carries more information as shown in Figure 2-12.

Word:	The	plane	lands	in	Lisbon
PoS-tag:	DET	NOUN	VERB	PROP	NOUN
Shape:	Xxx	xxxx	xxxx	xx	Xxxxxx

Figure 2-12: Example of multi-channel input for text classification (from http://davidbatista.net)

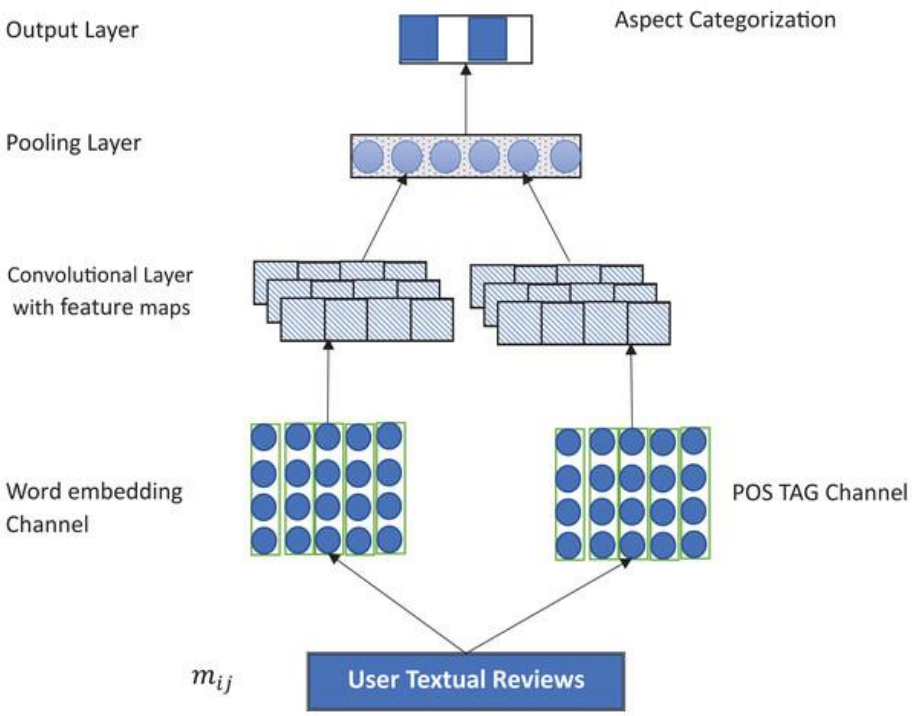


Figure 2-13: Architecture of multi-channel CNN for text classification (from Da'u & Salim, 2019)

In our study, we perform sentiment analysis with 5 classes and topic classification with 11 classes, softmax function is applied to the output layer, as we will describe in chapter 4.

2.8.5.2 Long short-term memory Neural Network

Long short-term memory Neural Network (LSTM) is a type of Recurrent Neural Network (RNN) that is made to overcome the vanishing gradient from a typical RNN by implementing self-loops and gating mechanisms (Goodfellow et al., 2017). Hence, it is worth to mention the concepts of RNN.

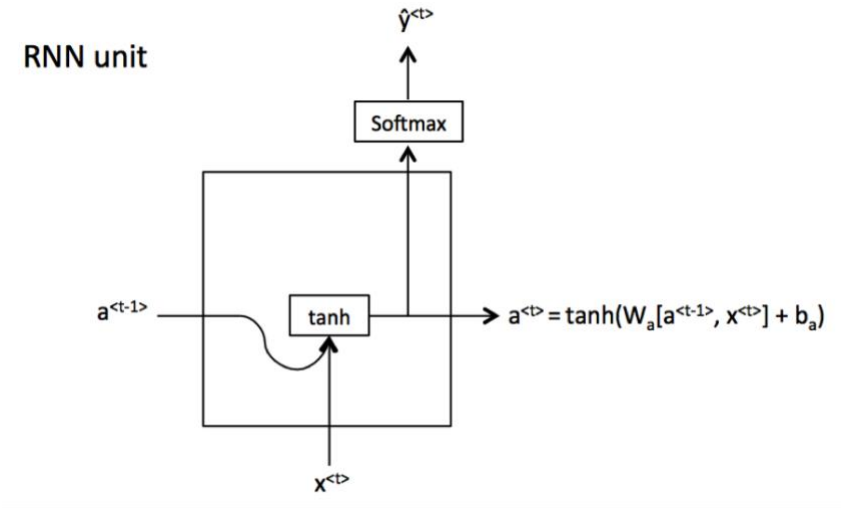


Figure 2-14: RNN unit (from Andrew, NG., 2018)

RNN is a type of sequence model that processes information in the form of timesteps. The notations of formulas are described in timestep manner. Each of the RNN units passes the activation value to the next timestep until the last timestep is reached as illustrated in figure **2-14**. At time t, RNN also incorporates the activation value from the previous timesteps ($a^{<t-1>}$) to predict the output ($a^{<t>}$) as opposed to only using the input ($x^{<t>}$) to predict (Andrew, NG., 2018).

For the sentence classification, we can apply Softmax classifier at last timestep for multi-class classification. There are many variations of RNN's e.g. many-to-one, many-to-many-one, bidirectional. The RNN architecture used in our study on sentence classification is called many-to-one as shown in Figure 2-15. In sentence classification problem, x might be text or customer review, for example, "*great bargain for the price*" and we represent each word as a word embedding vector of size 3. Figure The input x can be represented as follow.

Here, the length of input (T_x) is equal to 5, and

x^1 : "great" = [1,1,2]

x^2 : “bargain” = [1.2,0.1,2.1]

x^3 : “for” = [0.1,1,0.5]

x^4 : “the” = [1,2,3]

x^5 : “price” = [0.1,2.1,0.9] great bargain for the price

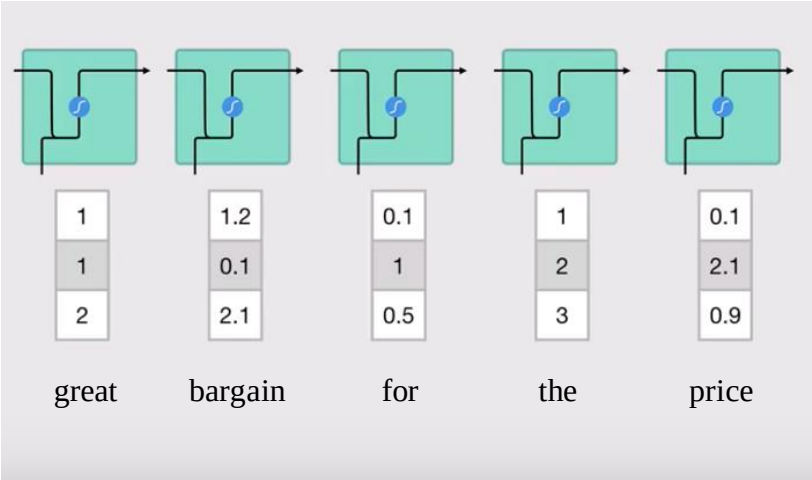


Figure 2-15: Example of input for sentence classification in RNN model (adapted from http://towardsdatascience.com)

An architecture of many-to-one RNNs with Softmax classifier at the last timestep to classify text is shown in Figure 2-16

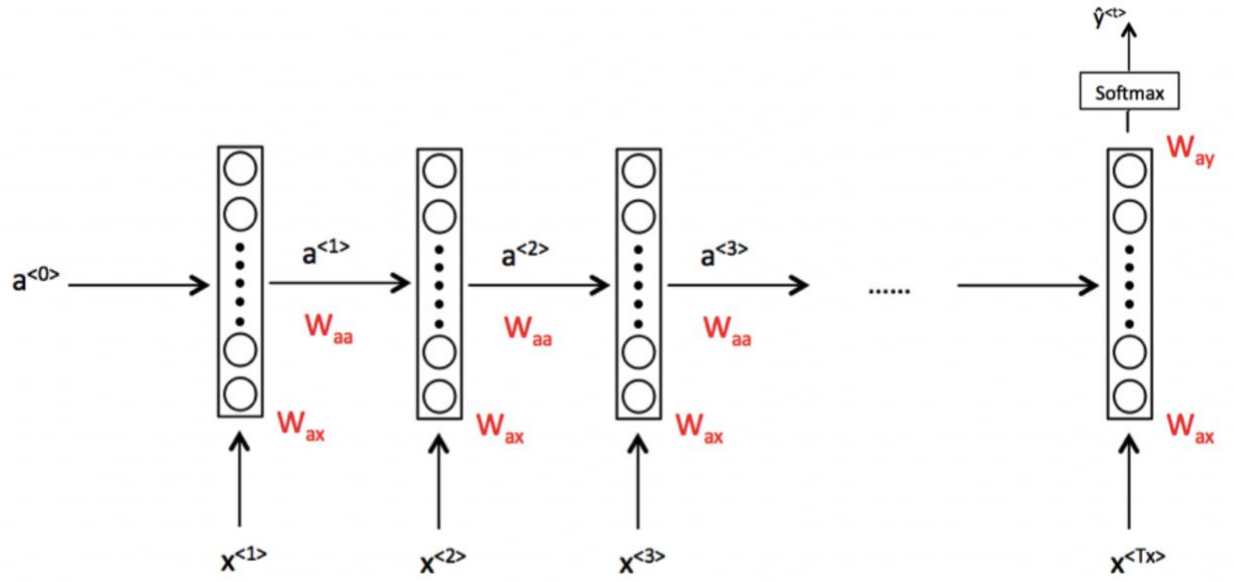


Figure 2-16: Architecture of many-to-one RNN with Softmax at the last timestep (from Andrew, NG., 2018)

where t represents the timestep, T_x is the length of input, a_0 is the time 0 activation which is a vector of zeros, $x^{<t>}$is the input, $\hat{y}$ is the predicted output, $a^{<t>}$is the predicted output. W_{ax} stands for the input parameters, W_{ax} is the activation parameters, W_{ax} is the for output prediction parameters. The parameter W_{ax} , W_{aa} , W_{ay} are the same in every timestep (Andrew, NG., 2018).

Long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997) introduces 3 gate mechanisms to the RNN architecture, namely, update gate, forget gate and output gate. Gate mechanism solves vanishing gradient which is an inherent problem for RNN. Gates allow LSTM to learn a very long range connection in a sequence and solve. Figure 2-17 shows the LSTM unit.

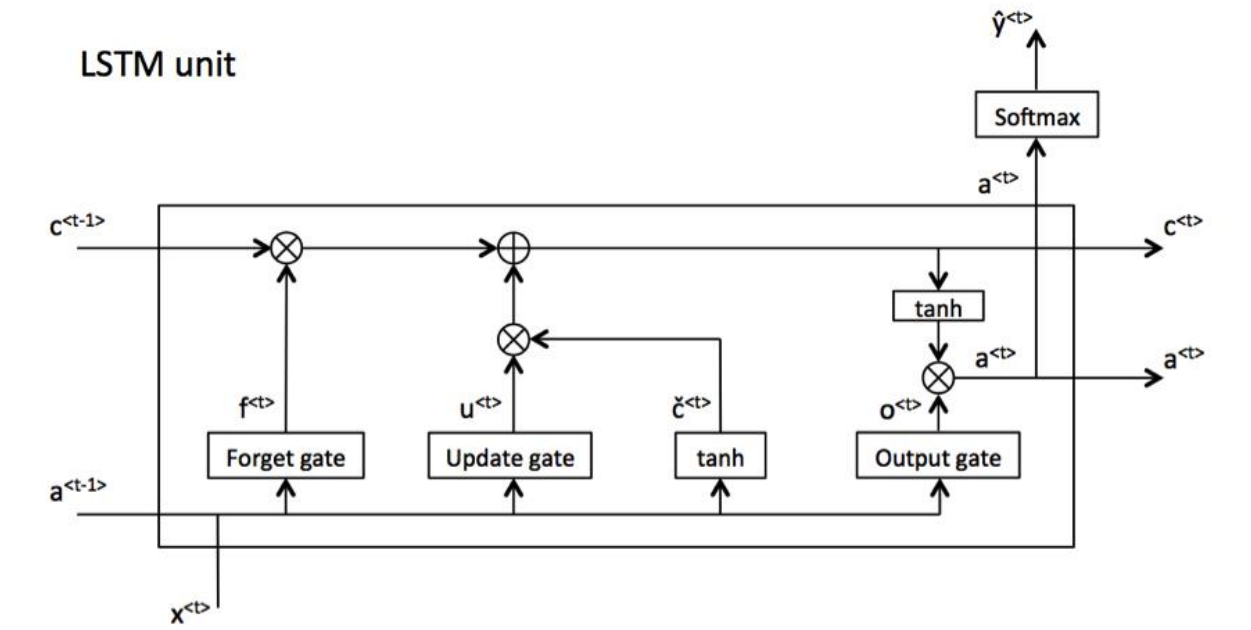


Figure 2-17: LSTM unit (from Andrew, NG., 2018)

The equations for LSTM unit at time t are described as follow:

$$\check{c}^{<t>} = \tanh\left(W_c[a^{<t-1>}, x^{<t>}] + b_c\right)$$

$$\text{update gate: } \Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$$

$$\text{forget gate: } \Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$$

$$\text{output gate: } \Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$$

$$c^{<t>} = \Gamma_u * \check{c}^{<t>} + \Gamma_f * c^{<t-1>}$$

$$a^{<t>} = \Gamma_o * \tanh\left(c^{<t>}\right)$$

where Γ_u is the update gate unit, Γ_f is the forget gate unit, Γ_o is the output gate unit,

b and W with subscripts u, f, o represents biases and weights for update, forget and output gates respectively, $c^{<t>}$ is a memory cell value, $\check{c}^{<t>}$ is a candidate value for replacing the memory cell, σ is sigmoid function, * is vector-vector elementwise multiplication, $x^{<t>}$ is the current input vector at time t, $a^{<t-1>}$is the output from the previous timestamp.

The main concept of LSTM is that using the activation from the previous timestamp, $a^{<t-1>}$ and the current input $x^{<t>}$ to compute all the gate values. The memory cell c provides memory to remember the value (Andrew, NG., 2018). LSTM pass the value c along from the first to the last timestep. That is the reason why LSTM is able to remember the words along the sentence. Figure 2-18 illustrates the sequence model of LSTM units.

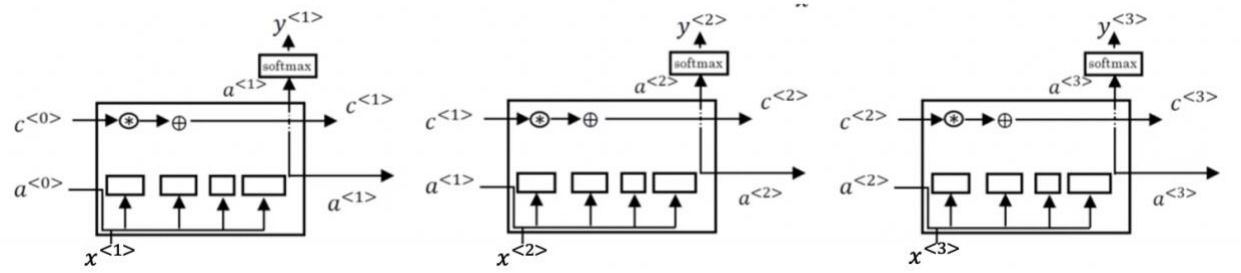


Figure 2-18: Sequence model of LSTM units (from Andrew, NG., 2018)

2.9 Rule-based Models for polarity and subjectivity

Rule-based approach for sentiment analysis aims to find keywords in the text and map each one to numerical scores or weights in a dictionary (Lane et al., 2019). Polarity of the text refers to degree of positiveness or negativeness of the text sentiment. For example, the polarity is quantified in the range [-1,1] where '-1' denotes extremely negative sentiment, '0' represents neutral sentiment and '1' represents extremely positive sentiment.

2.9.1 Valence Aware Dictionary for sEntiment Reasoning (VADER)

Valence Aware Dictionary for sEntiment Reasoning (VADER) is a rule-based model for general purposes in social media sentiment analysis tasks which was first proposed by Hutto et al., (2014). It is one of the most successful rule-based sentiment analysis algorithms (Lane et al., 2019). VADER takes "intensity" of a word into consideration instead of just using the polarity. For example, the words "good" and "excellent" would have the same score in a polarity-based

model. In valence-based approach like VADER model, the word "excellent" will be treated as more positive than "good". This parsimonious rule-based model incorporates a "gold-standard" sentiment lexicon that is tuned to micro-blog like contexts. The model leveraged the "wisdom-of-the-crowd" method to find a valid point estimate. Wisdom-of-the-crowd is the process of incorporating aggregated opinions from a collection of experts which often better than estimates from lone individuals(Hutto et al., 2014). Figure 2-19 explains the methods and process approach overview of VADER model.

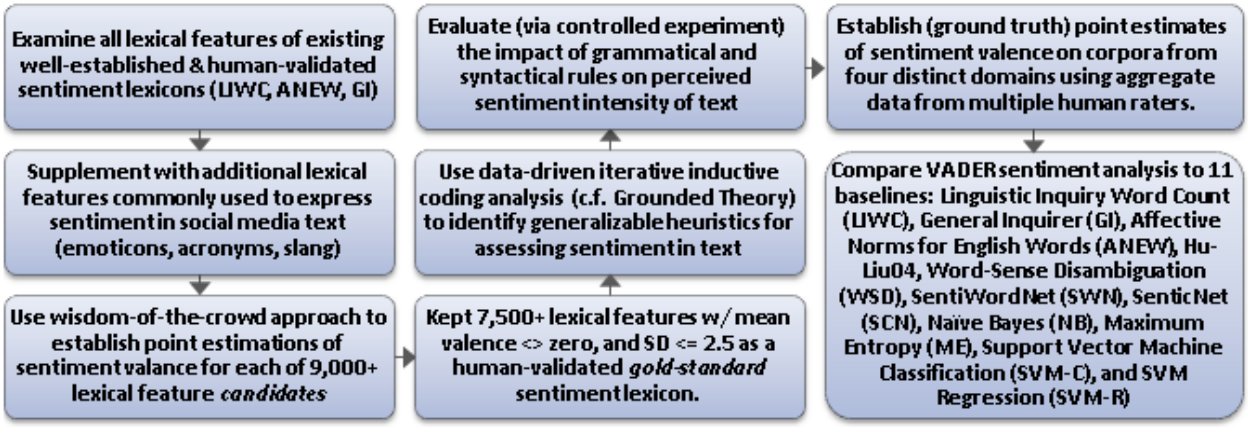


Figure 2-19: Methods and process approach overview (from Hutto et al., 2014)

The user interface used in obtaining the valid point estimates of sentiment valence (intensity) is shown in Figure 2-20.

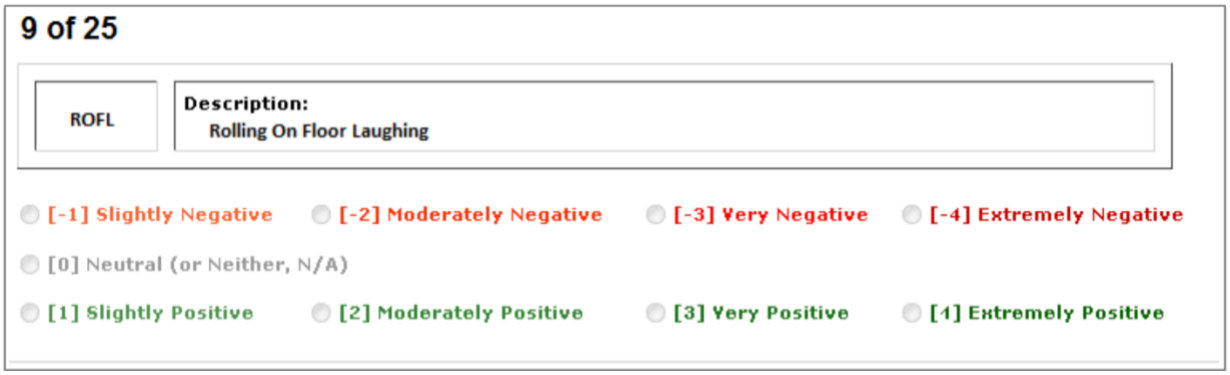


Figure 2-20: UI used in collecting sentiment valence (intensity) for VADER model

Ten independent human raters provided more than 90,000 lexical ratings in a scale of [-4,+4] using the Amazon Mechanical Turk (AMT) dataset. Only 7,500 lexical features survived from the criteria of having non-zero mean rating and having a standard deviation less than 2.5 (Hutto et al., 2014). The only drawback of VADER is that it considers only about 7,500 words. It does not look at all words in a document (Lane et al., 2019).

Chapter 3
Overview of the data

This section discusses in detail the datasets used. Two datasets used for this study are provided by Kaggle website. Amazon Fine Food Review dataset is used in sentiment analysis problem and US Consumer Finance Complaints dataset is used in text classification problem.

3.1 Dataset for sentiment classification problem

The Amazon Fine Food Review was collected and published by McAuley et al., (2013) in their research on online reviews. The dataset consists of online food reviews on website Amazon.com. Each review has a star rating representing the customer's sentiment towards the product and service ranging from 1 star to 5 stars. The counts of each review score are shown in Figure 3-1 below:

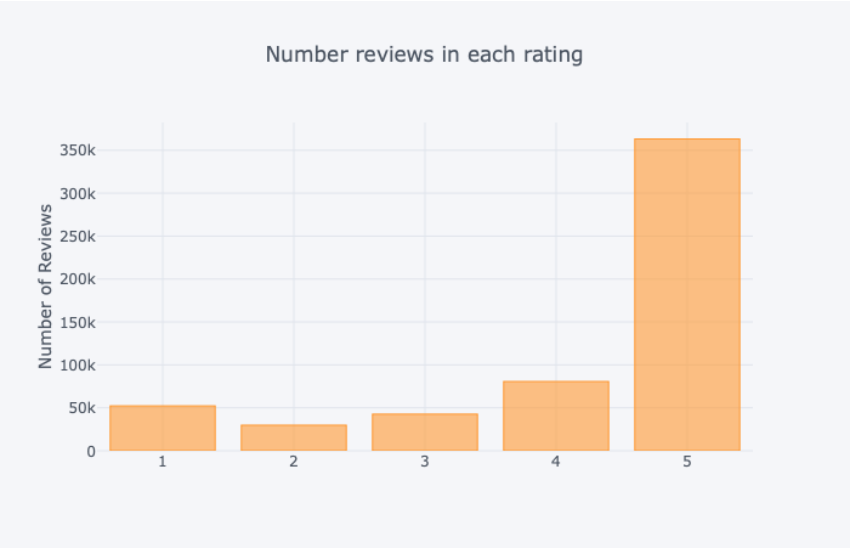


Figure 3-1: Frequency distribution of 'Review Score' attribute

The dataset has total of 568,454 reviews on 74,258 products. The Figure 3-1 shows 1-star: 52,268 observations, 2-star: 29,769 observations, 3-star: 42,640 observations, 4-star: 80,655

observations, and 5-star: 363,122 observations. The attributes and their descriptions are discussed in Table 3-1 below:

Table 3-1: Feature description for Amazon Fine Food Review dataset

Review attribute	Description	Variable type
Product ID	Unique identifier for the product	Categorical
User ID	Unique identifier for the user	Categorical
Profile Name	Profile of the user	Text
Helpfulness Numerator	Number of users who found the review helpful	Numerical
Helpfulness Denominator	Number of users who voted whether the review was helpful or not	Numerical
Score	Rating between 1 and 5	Ordinal
Time	Timestamp of the review	Numerical
Summary	Brief summary of the review	Text
Text	Text of the review	Text

3.1.1 Overview of the texts for sentiment classification problem

The review text contains a detailed description of the product from the user's perspective. Moreover, it also describes the overall sentiment of the user toward the product apart from the description of the product itself. Some of the examples of the review text are shown below:

1) *5-star review text: "That's exactly what I was looking for to bake some chouquettes (puffs with small sugar pearls on top). Works great, yum!!"*
2) *4-star review text: "I only use raw sugar, it did seem a little smaller than the normal crystals but it is still good. Will buy again."*

3) *3-star review text: "I was amazed at how quickly it arrived! It is a cute stocking stuffer, but only OK as far as user appeal. You get what you pay for."*
4) *2-star review text: "I really love Kettle brand chips, but these are rather disappointing. There is very little sour cream or onion flavor."*
5) *1-star Review text: "These condiments are overpriced and terrible. The classic is disgustingly sweet. The spiced tastes like a bad spicy marinara sauce from a chain restaurant."*

Besides, the summary text represents a user's sentiment in a confined manner. The information is conveyed in a few words in this case. Some of the examples of the summary text are shown below:

1) *5-star review summary: "Best deal ever!"*
2) *4-star review summary: "Great Bargain for the Price"*
3) *3-star review summary: "Not the greatest tasting."*
4) *2-star review summary: "If you can't handle caffeine, this is not for you."*
5) *1-star review summary: "Big disappointment"*

Table 3-2: Statistics of document lengths for Amazon Fine Food Review

Statistic	Value
Mean	80.25
SD	79.42
Min	1
25^{th} percentile	33
50^{th} percentile	56

75th percentile	98
Max	3,432

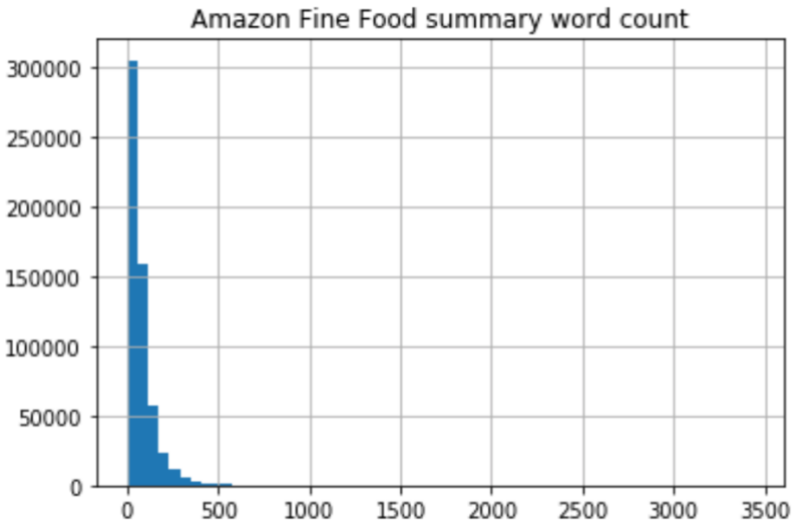


Figure 3-2: Distribution of document lengths for Amazon Fine food Review

Table 3-2 shows the statistics of document lengths, and the histogram in Figure 3-4 shows the distribution of document lengths for Amazon Fine Food Review.

In this study, review text is the only raw data input for predicting the sentiment. Including the review summary might improve the prediction accuracy. However, this feature is not provided in all online and social media platforms.

3.2 Dataset for topic labeling problem

The Consumer Finance Complaints dataset is available on the website Kaggle.com. The data is provided by Consumer Financial Protection Bureau (CFBP) who collects consumers' complaints about products and services to the companies in the financial industry such as Bank of America,

Capital One, Paypal Holding, etc. The aim of this dataset collection is to improve the financial marketplace. In this study, we focus on text classification algorithms to classify the topic of the complaints.

3.2.1 Overview of the texts for topic labeling problem

Table 3-3: Feature description for Consumer Finance Complaints dataset

Complaint attribute	Description	Variable type
Product	Topic of the complaint	Categorical
Sub Product	Subcategory of the complaints topic	Categorical
Issue	Brief summary of the complaints	Text
Consumer Complaint Narrative	Text of the complaint	Text
Company	Financial company which the consumer filed the complaint to	Text
State	State address of the consumer	Text
Date sent to Company	The date on which the complaint was sent	Numerical
Submitted Via	The method user submitted the complaint	Categorical
Company Response to Consumer	Status of the response	Categorical
Complaint ID	Unique identifier for the complaint case	Text

Table 3-4: Categories and counts of the Products in Consumer Finance Complaints dataset

Product	Observations

Debt Collection	17,552
Mortgage	14,919
Credit Reporting	12,526
Credit Card	7,929
Bank Account or Service	5,711
Consumer Loan	3,678
Student Loan	2,128
Prepaid Card	861
Payday Loan	726
Money Transfers	666
Other Financial Service	110

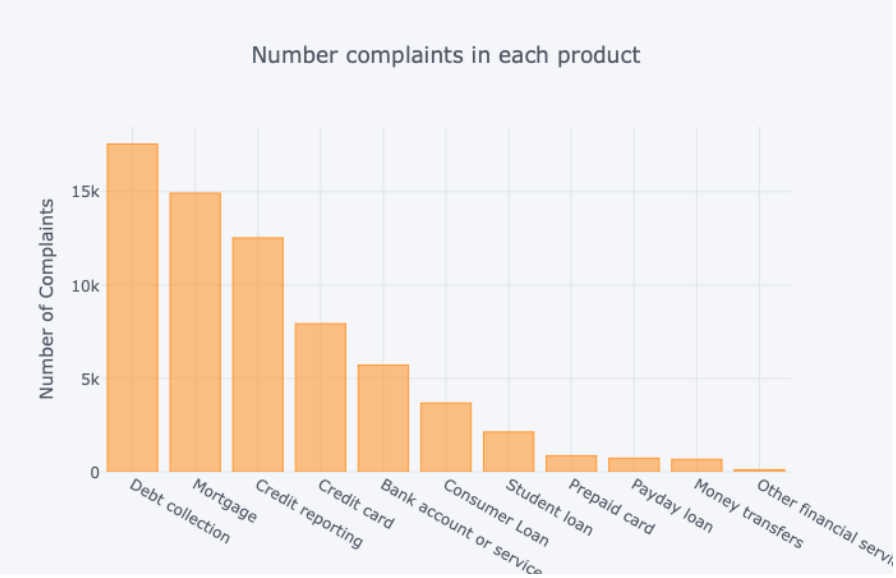


Figure 3-3: Frequency distribution of 'Product' attribute

The dataset consists of 66,806 observations. In the study on this dataset, we employ various text classification models to predict 11 topics of the complaints. The data in the column "Consumer Compliant Narrative" are fed into text classification models to predict the 11 compliant topics in the "Product" column. The description for each attribute is shown in table 3-3. The counts for each class can be found in table 3-4 and the distribution of the classes are shown in Figure 3-3. Some of the examples of the review text are shown below:

1) *Debt Collection topic: "I resided at XXXX from XXXX (see attached), and I had a move out balance of {$110.00}. The apartment collection agency (XXXX) posted this on my credit report TWICE. The balance has been paid to XXXX since XXXX XXXX, XX/XX/XXXX (see attached), however it is still showing up on all credit bureau as a DEBT and as a DUPLICATE. The collection agency is giving me the run around and it has affected me tremendously, and I need this debt to be removed IMMEDIATELY!"*
2) *Credit Reporting topic: "There is a XXXX XXXX account on my credit report that does not belong to me. I have attem have attempted to dispute several times to no avail."*
3) *Prepaid Card topic: "I was offered a pre-paid card from Continental financial, but was not interested in their services. They made an inquiry into my credit which has negatively affected my credit. They will not remove the inquiry and it has negatively impacted my credit. "*

Table 3-5: Statistics of document lengths for US Consumer Finance Complaints

Statistic	Value
Mean	190.64

SD	166.83
Min	1
25th percentile	71
50th percentile	136
75th percentile	254
Max	1,284

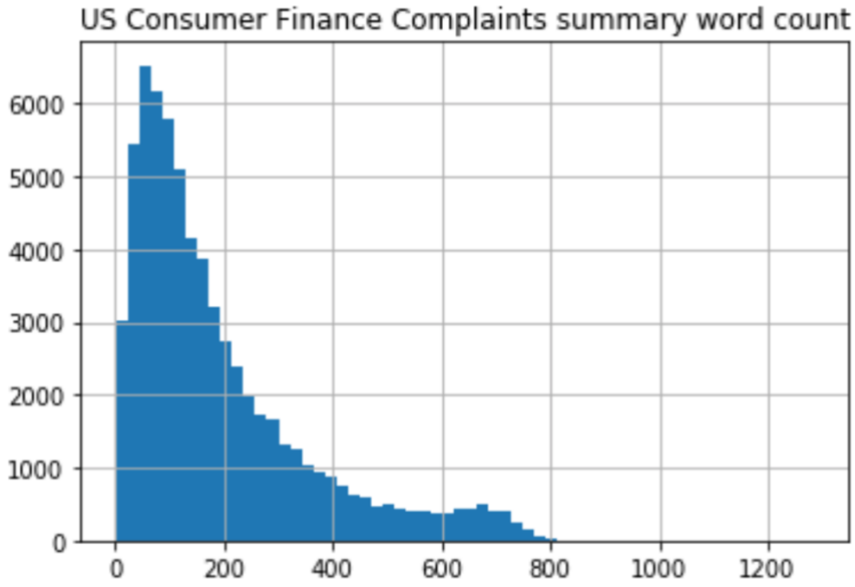


Figure 3-4: Distribution of document docu,ent lengths for US Consumer Finance Complaints

A document length refer to the total length of word sequences including all sentences in a document. Table 3-5 shows the statistics of document lengths, and the histogram in Figure 3-4 shows the distribution of document document lengts for US Consumer Finance Complaints dataset.

In this study, review text is the only raw data input for predicting the sentiment. Including the review summary might improve the prediction accuracy, however, this feature is not provided in all online and social media platforms.

This dataset is class-imbalanced. The minority class is greatly outnumbered by the majority class. If the goal is to also label the minority class correctly, using only micro-averaged accuracy is inappropriate due to the insensitivity to the imbalance of the classes. Macro-averaged accuracy is crucial in selecting the best model for this dataset. The best accuracy measures will be further discussed in the Model Validation section in Chapter 4.

Chapter 4
Methodology

4.1 Workflows for building automated text analysis models

This section discusses the process of text classification. When solving a machine learning problem, we need to follow a specific workflow which varies with different models. We will discuss the workflows for traditional models, deep learning models, and rule-based sentiment models.

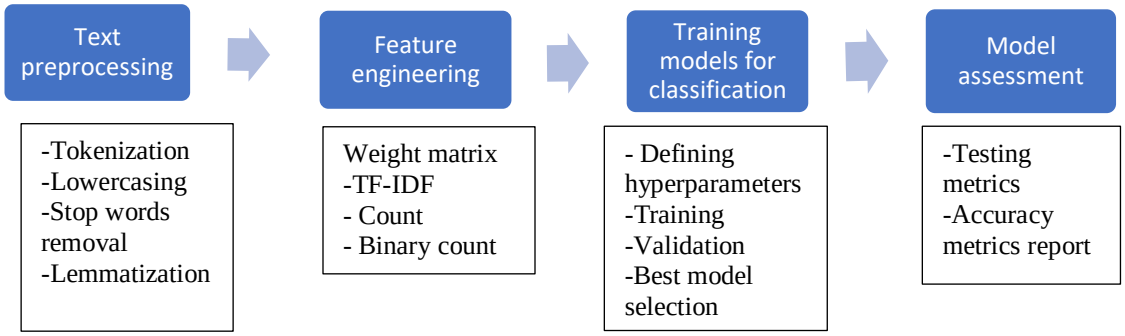


Figure 4-1: Workflow for conventional models (adapted from Olson et al., 2016)

Figure 4-1 illustrates the workflow for traditional models namely, K-nearest neighbors, Multinomial Naïve Bayes, Logistic Regression and Support Vector Machine. It encapsulates the following steps:

1) Performing text preprocessing steps to remove noisy and unimportant terms
2) Performing document vectorization to prepare weight matrix for machine learning models
3) Training the models with different sets of hyperparameters and selecting the best model on the validation scores
4) Performing model assessment with test dataset and gives the accuracy metrics

After data preprocessing in step 1 and document vectorization in step 2, the sparse matrix with the size of [number of sentences × number of unique tokens] will be passed to a machine learning

algorithm. Next, we need to try different sets of hyperparameters for that algorithm in step 3. The validation process is done in this step to find the best model for each algorithm. In our study, the best model is selected by the criteria such as accuracy or macro F-1 score. Finally, in step 4, the accuracy of the best model will be evaluated with the test dataset and show the accuracy metrics.

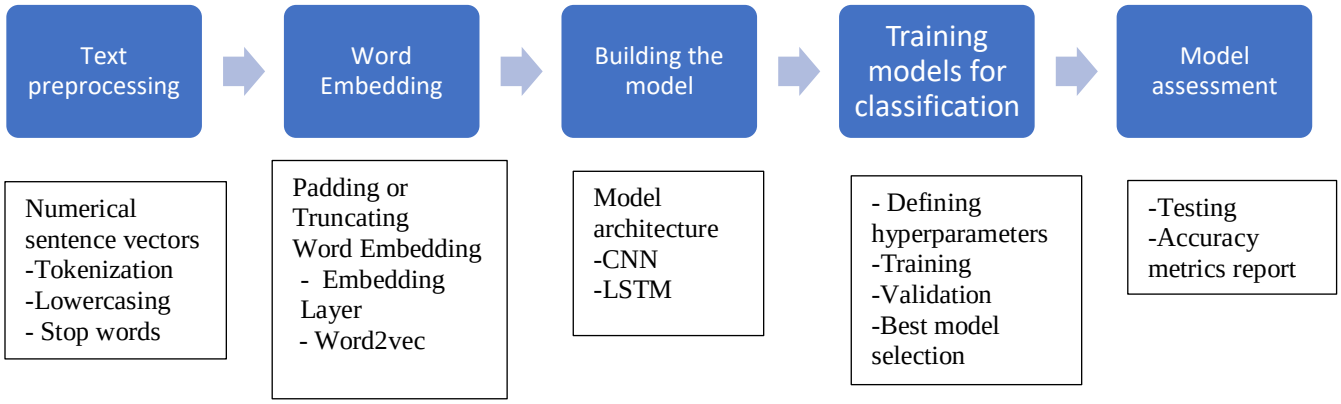


Figure 4-2: Workflow for deep learning models (adapted from Olson et al., 2016)

Figure 4-2 illustrates the workflow for traditional models namely, 1-D Convolutional Neural Network (CNN), Long Short-term Memory Neural Network (LSTM). It encapsulates the following steps:

1) Performing text preprocessing to prepare for numerical sentence vectors generation
2) Performing word embedding to convert each word in the sentence be a vector (dense representation)
3) Building the deep learning models
4) Training the models with different sets of hyperparameters and selecting the best model based on the validation scores

5) Performing model assessment with test dataset and gives the accuracy metrics

In step 1, we apply the function to generate numerical sentence vectors by tokenization. In this step, the vectors are not the same length. Next, padding or truncating is done to preset the maximum sentence length of each sentence vector. The deep learning toolkits such as Keras, PyTorch, MXNet have built-in functions to perform tasks in step 1.

In step 2, the word embedding model transforms the sentence vectors matrix into the embedded sentence vector matrix. Basically, it transforms the size of the data from [number of sentences × length of sentences] to be [number of sentences × length of sentences × output dimension]. Note that the output dimension refers to the embedding dimension of each token (word).

In step 3, deep learning models are created. As an example of Keras library, for CNN model, we first need to specify number of filters, maximum length of the sentence and padding method, activation function, strides, and input shape in order to initiate the input of the 1D convolutional neural net model. Then, we need to specify the pooling layer and fully connected layer, and output layer (Lane et al., 2019). The output layer (e.g. Softmax classifier) has the number of nodes equal to the number of target classes in our problem.

In the case of LSTM's, first we need to specify the maximum length of the sentence and padding, number memory cells in each LSTM unit in this sequence model and the output layer (e.g. Softmax classifier)

In step 4, as the same ideas of dropout, epoch size, number epoch, number of iterations are common in optimizing and prevent overfitting of the neural network models, we need to specify those hyperparameters before training. Dropout rate can mitigate the risk of overfitting while allowing the model to have complexity as it needs to match the data, 20% to 50% dropout rate is a safe range for NLP problems using recurrent networks (Lane et al., 2019).

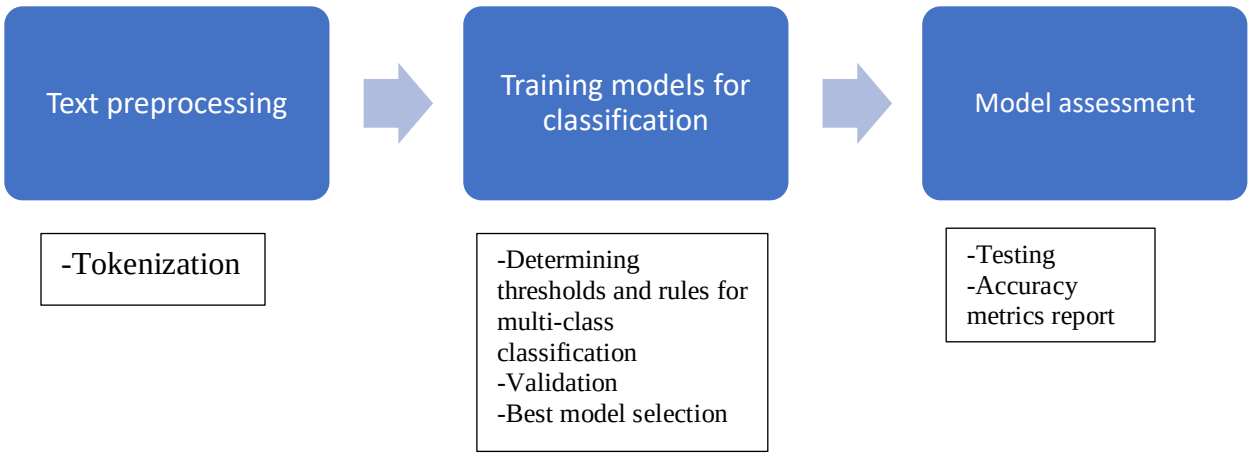


Figure 4-3: Workflow for VADER models in sentiment classification problem

Figure 4-3 illustrates the workflow for VADER models in sentiment classification problem. It encapsulates the following steps:

1) Performing tokenization in text preprocessing step
2) Performing document vectorization to prepare weight matrix for machine learning models
3) Training the models with different sets of hyperparameters and selecting the best model on the validation scores

VADER is tested only on Amazon Fine Food review dataset. In step 1, the text should not be stemmed or lemmatized because VADER has different ratings depending on the form of the word. In step 2, we set the different sets of rules and thresholds for the output scores (compound, positive and negative) to classify the document into 5 categories (5-star rating). The validation process is done in this step to find the best model. In step 4, the accuracy of the best model will be evaluated with the test dataset and show the accuracy metrics.

4.2 Exploratory data analysis

We have discussed the distribution of 'Review Score' attribute for Amazon Fine Food review dataset and the distribution of 'Product' attribute for US Consumer Finance Complaint dataset. This section mainly discusses the distribution of the scores that VADER model gives on Amazon Fine Food texts.

Since VADER gives scores in continuous values but the target of our Amazon Fine Food review consists of 5 categorical values (1 star to 5 stars), it is necessary to explore the distribution of the scores VADER gives. This step will help determine the thresholds and rules to set to VADER model to classify the document into the 5-star rating. The positive, negative and compound scores are analyzed from VADER model. The distribution of the compound scores are shown below in Figure 4-4, Figure 4-5, Figure 4-6, Figure 4-7

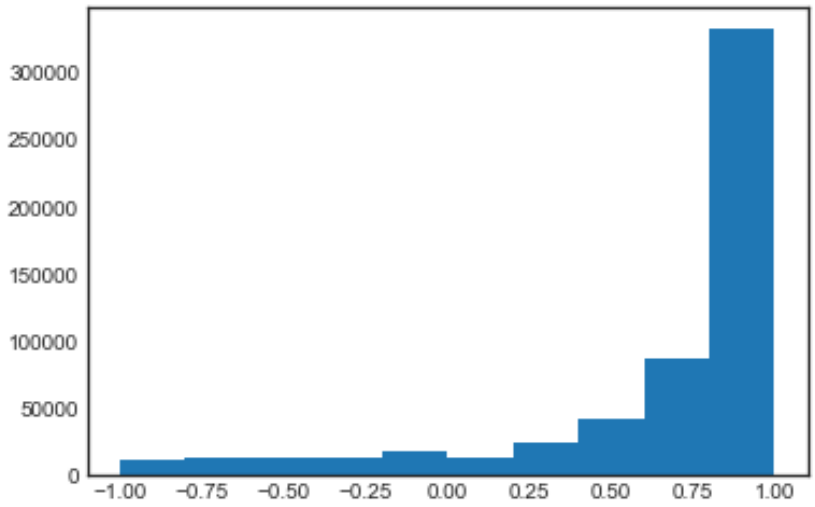


Figure 4-4: Distribution of compound scores from VADER on Amazon dataset

Figure 4-5: Normalized distribution of compound scores from VADER on Amazon dataset

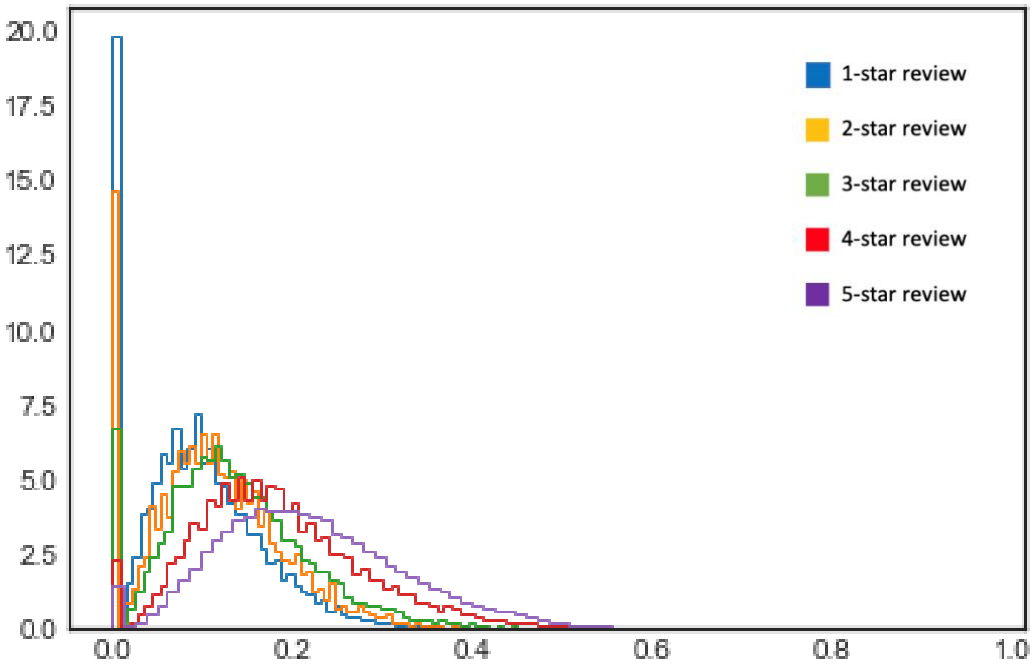


Figure 4-6: Normalized distribution of positive scores from VADER on Amazon dataset

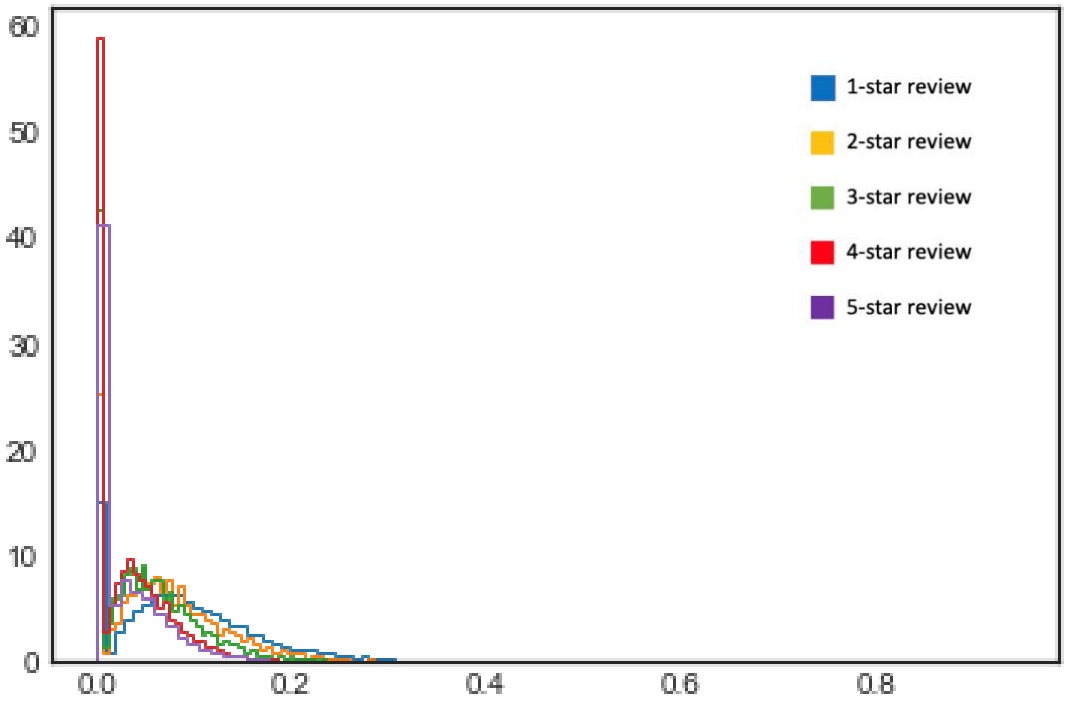


Figure 4-7: Normalized distribution of negative scores from VADER on Amazon dataset

Figure 4-4 shows that the majority compound scores VADER predicted on Amazon documents have high compound sentiment scores, which we can infer that there are a lot of 5 -star comments. At first glance, this looks similar to the Frequency distribution of 'Review Score' in Figure 3-1. However, when we drill down to each target variable (1-star to 5-star) and plot the histogram of the scores from VADER, it shows that it is not always the case that 1-star review will have low compound score, low positive scores, and high negative scores, as shown in Figure 4-5, Figure 4-6, Figure 4-7. Therefore, it is hard to set the thresholds to classify every class accurately. The results in the next chapter show that VADER only performs well in the target "5-star" target, the rest shows low F-1 score compared to other algorithms.

4.3 Text preprocessing

Text preprocessing is done to eliminate the inconsistent data and noise. It helps in maximizing the classifier performance. In many cases, it helps in improving the accuracy and avoiding the overfitting problem by eliminating the noisy features.

4.3.1 Tokenization

Tokenization is the process of splitting the sentences into words. Each word is later considered as a separate token (Hotho et al., 2005). It simply divides a sentence into a list of words (tokens).

4.3.2 Converting uppercase tokens into lowercase

In this step, the uppercase words are converted to lower case to avoid potential duplication of the words since they have the same meaning. This helps in reducing the dimensionality of the feature set.

4.3.3 Removal of punctuation marks

In this step, the punctuation marks are removed from the text as they do not provide any extra information while extracting the semantics from the text (Hotho et al., 2005).

4.3.4 Stop words removal

Stop words are words that occur at high frequency but hardly convey useful information. In traditional sentiment classifiers. For traditional classifiers, removing stop words results in higher accuracy compared to using unprocessed dataset (Ghag et al., 2015). Stop words are frequently used words which do not convey much meaning (Hotho et al., 2005). Table 4-1 shows an example of stop words removal

Table 4-1: Examples of stop words removal

Sample text with stop words	Sample text without stop words
There is a tree near the river	There tree near river.
A swimmer likes swimming, thus he swims.	Swimmer likes swimming swims.
We are good friends.	We good friends.

4.3.5 Lemmatization

Lemmatization is a text normalization process of reducing the inflectional form of words into base or dictionary form which is called ‘lemma’. For example, lemmatization converts the word ‘ran’ into its base form ‘run’. In lemmatization, complete morphological analysis of words is done to ensure that the base word belongs to the dictionary. The root word in stemming is not required to be a valid word from the language (Manning et al., 2008).

4.4 Model training and validation

“*The training set is used to fit the models; the validation set is used to estimate prediction error for model selection; the test set is used for assessment of the generalization error of the final chosen model. Ideally, the test set should be kept in a “vault,” and be brought out only at the end of the data analysis.”* (Hastie et al., 2017).

In this step, we train the models with different sets of hyperparameters and select the best model based on the highest macro F-1 validation scores. For Naïve Bayes, and Support Vector Machine, the model in each of the hyperparameters combination in grid-search space will be tested 5 times using k-fold cross validation. For deep learning models i.e.—CNN, RNN-LSTM, the simple hold-out method was preferred over the cross-validation method due to the time and computational constraints. For deep learning models, the portion of target classes is retrained to be the sameas that of the original dataset. For Amazon Fine Food review dataset, the dataset is split into 3 parts, i.e.— train set, validation set, test set in the ratio of 70%, 20%, 10%, respectively. For US

Consumer Financial Complaint dataset, the dataset is split into 3 parts, i.e.— train set, validation set, test set in the ratio of 70%, 10%, 20%, respectively. The reason we keep the test size 20% is because the "Other Financial Service" which is the smallest class consists of only 110 observations, therefore, we need a higher portion for test set to represent this class. For conventional models, the proportion of the test set is the same as mentioned above to make the experiment consistent. The difference is that this time, for conventional models, we perform 5-fold validation in training and validation process to make it more robust instead of just using simple hold-out method.

4.4.1 One vs Rest strategy

We will also experiment the "one-versus-rest" strategy for the topic labeling problem. In this approach, given that we have r target levels, instead of performing multi-class classification, we can create r models each of the models is good at recognizing one pattern from all the others. Basically, we make r models each of which is a binary classifier specifically attuned to predict each class (Kelleher, et al. 2015). This method will be employed on Logistic Regression, KNN and Naïve Bayes models to compare the performance against multi-class classification. Although this method is popular in predicting multi-label response which means one document can have more than one label, this method can still be applied in our study of multi-class single label problem. The strategy used in our study is to classify a document starting from using model 1, if the model 1 classify as "False", it will proceed to use model 2, otherwise, it will stop. This process passes along until model r if none of the prior models classify the response as "True".

4.5 Model evaluation

This section discusses the process of model evaluation and the accuracy metrics chosen in our study.

From the training and validation step, we came up with the best model. In this step, we evaluate that model by performing classification on the test dataset and show the accuracy metrics.

4.5.1 Accuracy metrics

Both Amazon Food Review dataset and US Consumer Finance Complaints dataset are class-imbalanced. For the unbalanced datasets, simply showing the accuracy from the formula $\frac{(tp+tn)}{(tp+tn+fp+fn)}$ can mislead the performance.

F-1 score, which is the harmonic mean of precision and recall, is a more appropriate measure in reporting the accuracy metrics multi-class problem that has an imbalanced dataset. F-1 for each target class evaluates how likely can we predict each class correctly. $F\text{-}1_{micro}$, $F\text{-}1_{macro,}$ and $F\text{-}1_{weighted}$ are also shown in the analysis section. All the metrics mentioned above can be calculated from the confusion matrix which is shown in Figure 4-6.

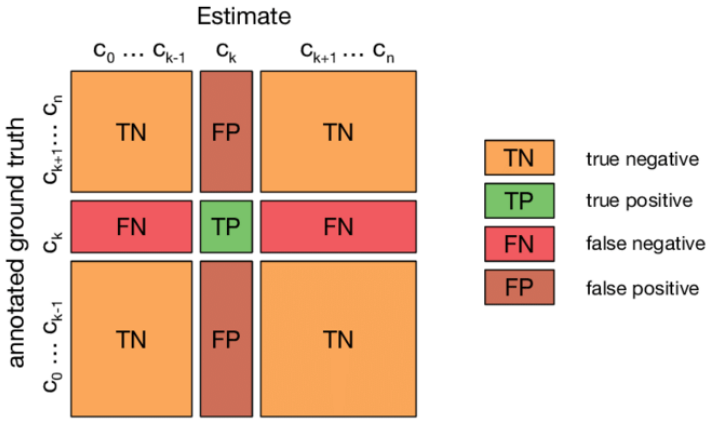


Figure 4-8: Confusion matrix for multi-class classification (from http://researchgate.net)

Given that the observing class is class C_k. Each column represents the predicted label, whereas each row represents the actual label.

The parameters of the confusion matrix can be described as below:

1) True positive (TP) – sample belonging to positive class predicted as positive
2) True negative (TN) – sample belonging to negative class predicted as negative
3) False positive (FP) – sample belonging to negative class predicted as positive
4) False negative (FN) - sample belonging to positive class predicted as negative

Selecting the best model is subjective. Macro-averaging treats all classes equally while micro-averaging favors bigger classes (Sokolova et al., 2009). In some cases, the model which gives highest F-1 score in particular classes is preferred over the model with the best overall F-1 score or best overall accuracy. For example, in sentiment classification problem for customer review, correctly detecting the unhappy clients and having quick responses can prevent the companies from losing businesses. Thus, the F-1 score for the "very bad" sentiment (review = 1 star out of 5 stars) is one of the most crucial scores to select the best model to be deployed in the real application.

Another example from topic labeling problem, the best model can be specifically attuned to classify some consumer complaints topics correctly instead of having the highest overall F-1 score or overall accuracy. This depends on urgency or importance each topic have an impact on the company's business. Table 4-2 lists and explains the accuracy measures where the subscript M refers to macro-averaged measure.

Table 4-2: Performance measures for classification (from Sokolova et al., 2009)

Measure	Formula	Evaluation focus

Accuracy	$\frac{(tp + tn)}{(tp + tn + fp + fn)}$	Overall effectiveness of a classifier
Precision	$\frac{tp}{(tp + fp)}$	Class agreement of he data labels with the positive labels given by the classifier
Recall (Sensitivity)	$\frac{tp}{(tp + fn)}$	Effectiveness of a classifier to identify positive labels
F-score	$\frac{(\beta^2 + 1) \cdot tp}{(\beta^2 + 1) \cdot tp + \beta^2 \cdot fn + fp}$	Relation between data's positive labels and those given by a classifier
Specificity	$\frac{tn}{(fp + tn)}$	How effectively classifier identifies negative labels
AUC	$\frac{1}{2}(\frac{tp}{tp + fn} + \frac{tn}{tn + fp})$	Classifier's ability to avoid false classification
Precision$_M$	$\frac{\frac{\sum_{i=1}^{l} tp_i}{\sum_{i=1}^{l}(tp_i + fp_i)}}{l}$	Agreement of the data class labels with those of a classifiers if calculated from sums of per-text decisions
Recall$_M$	$\frac{\frac{\sum_{i=1}^{l} tp_i}{\sum_{i=1}^{l}(tp_i + fn_i)}}{l}$	Effectiveness of a classifier to identify class labels if calculated from sums of per-text decisions
F-score$_M$	$\frac{(\beta^2 + 1) \cdot Precision_M Recall_M}{(\beta^2) \cdot Precision_M + Recall_M}$	Relations between data's positive labels and those given by a classifier based on a per-class average

Chapter 5
Analysis and Results

This section discusses the analysis of sentiment classification on Amazon Fine Food reviews dataset and topic labeling on US Consumer Finance Complaints dataset. The models and feature engineering methods for this study are shown below in Table 5-1.

Table 5-1: Models used in text classification problems

	Sentiment classification	Topic labeling	Feature engineering
K-nearest Neighbors (KNN)	Yes	Yes	TF-IDF
Multinomial Naïve Bayes	Yes	Yes	TF-IDF
Support Vector Machine (SVM)	Yes	Yes	TF-IDF
Logistic Regression	Yes	Yes	TF-IDF
Convolutional Neural Network (CNN)	Yes	Yes	Embedding Layer
Long short-term memory (LSTM) recurrent neural network	Yes	Yes	Embedding Layer
Valence Aware Dictionary and sEntiment (VADER)	Yes	No	-

The preprocessing methods such as stop words removal, lowercasing, punctuation removal, and lemmatization are not included in the table 5-1 since they are treated as same as the hyperparameters in this study. In topic labeling, the one-versus-rest strategy will also be employed in the conventional models like KNN, Naïve Bayes and Logistic Regression.

5.1 Analysis of sentiment analysis problem

The first step in the analysis is to perform text preprocessing. The vectorized features are prepared by TF-IDF method for KNN, Naïve Bayes, Logistic Regression and SVM. The data is selectively preprocessed by stop words removal, lowercasing, punctuation removal, and lemmatization before generating weight matrix using TF-IDF, whereas the Embedding Layer is performed to generate

embedded sentence vector matrix for CNN and LSTM. For VADER, we set the rules and thresholds for the model to perform multi-class classification.

The original dataset is split into train set , validation set and test set with the percentage 70%, 20% , 10% respectively for deep learning models. For the conventional models, we set the size of the test data to be 10% to make it consistent with the deep learning method. And the rest 90% will be used in training and validation process. 5-fold validation is used for the conventional models to make the result more robust than just using the simple hold-out method.

The accuracy for each model on Amazon Fine Food Review dataset is shown Figure 5-1 and Table 5-2.

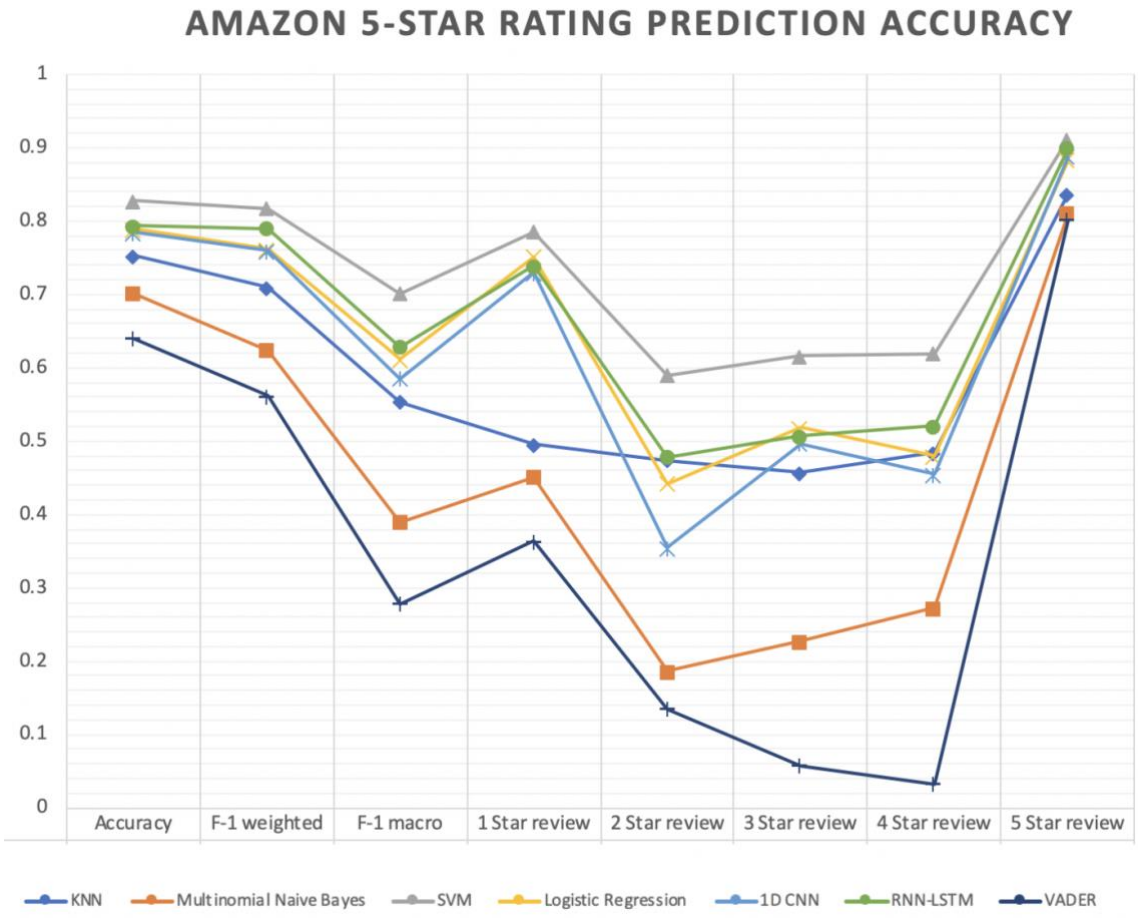


Figure 5-1: Performance comparison of classifiers for Amazon Fine Food Review dataset

				F-1 Score				
Model	Accuracy	F-1 weighted	F-1 macro	1 Star review	2 Star review	3 Star review	4 Star review	5 Star review
KNN	0.7526	0.7089	0.5532	0.4956	0.4735	0.457	0.4836	0.8374
Multinomial Naive Bayes	0.7014	0.6240	0.3896	0.4511	0.1864	0.2273	0.2722	0.8111
SVM	0.8283	0.8176	0.7010	0.7858	0.5895	0.6157	0.6198	0.9123
Logistic Regression	0.7891	0.7623	0.6112	0.751	0.4419	0.5180	0.4791	0.8837
1D CNN	0.7844	0.7592	0.5855	0.7295	0.3543	0.4966	0.4543	0.8878
LSTM	0.7932	0.7897	0.6288	0.7384	0.4791	0.5064	0.5204	0.8996
VADER	0.6397	0.5618	0.2783	0.3635	0.1355	0.0577	0.0332	0.8019
Supports	56846	56846	56846	5224	3045	4186	8187	36204

Table 5-2: Accuracy metrics from sentiment classification on Amazon Fine Food Review

Among all the models used in sentiment analysis, the Support Vector Machine (SVM) scored highest in both macro F-1 Score and accuracy. This result is in accordance with Aggarwal et al., (2012) regarding high suitability of conventional classifiers such as SVM for sparse text classification problems. The SVM model shows 82.83% overall accuracy and 70.10% of macro F-1 score. LSTM achieved the second best overall accuracy and macro F-1 score.

The third highest accuracy is from a conventional model, Logistics Regression. It yields an accuracy of 78.91% and a macro F-1 score of 61.12%.

Comparing the deep learning models, LSTM performs better than 1D Convolutional Neural Network (1D ConvNet). LSTM yields an accuracy of 79.32% and a macro F-1 score of 62.88%, whereas 1D ConvNet shows an accuracy of 78.44% and a macro F-1 score of 58.55%. The results of deep learning models shown in table 5-2 are from the models that used 100-dimensional word embedding vectors that are fitted from our own courpus using “Keras Embedding Layer”.

Stanford GloVe pre-trained word embedding was attempted on both 1D CNN and LSTMs but did not show significant improvement from such baseline deep learning models, therefore, it was not included in here.

Multinomial Naïve Bayes performed poorly in the small classes. The study on short text classification with a maximum length of 200 words from Song et al., (2014) stated that Bayes model generally face difficulty to obtain high accuracy when the labeled information is insufficient. Our sentiment analysis was performed on documents with an average length of 80 words which is longer but shows a similar result for Naïve Bayes.

VADER scored low in overall accuracy, the only class that VADER can capture well is the "5-star" class which represents extremely positive sentiment with 80.2% in F-1 score in that particular class. It is hard to distinguish the other classes. For example, it is not always the case that 1-star review will have a low compound score, low positive scores and high negative scores shown in VADER exploratory data analysis in (please refer to section Section 4.1). Embedding Layer (Keras API) and Stanford GloVe was used for 1D CNN and LSTMs.

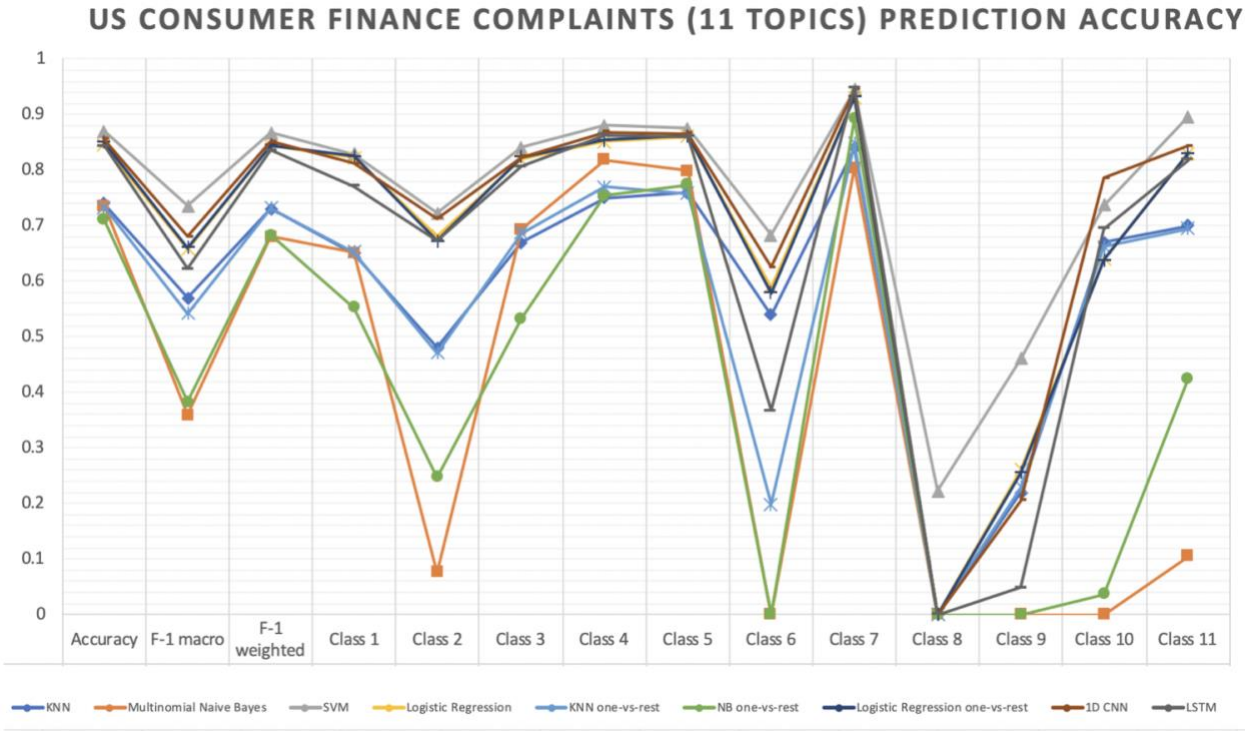
The selection of hyperparameters for the machine learning models in this study is shown in Table 5-4.

5.2 Analysis of topic labeling problem

The process for the analysis of topic labeling problem are the same as explained in Section 5.1 except that the data used in this problem is US Consumer Finance Complaints dataset whose target consists of 11 classes (topics of the consumer complaints) and the original dataset is split into train set, validation set and test set with the percentage 70%, 10% , 20% respectively for deep learning models. For conventional models, we set the size of the test data to be 20% to make it consistent with the deep learning method. And the rest 80% will be used in 5-fold validation method.

Besides, the VADER model is not used for this topic labeling problem since VADER is specifically made for sentiment analysis tasks. The accuracy for each model on US Consumer Finance Complaint dataset is shown in Figure 5-2.

Figure 5-2: Performance comparison of classifiers for US Consumer Finance Complaints



				F-1 Score										
Model	Accuracy	F-1 macro	F-1 weighted	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	Class 10	Class 11
KNN	0.7432	0.5738	0.7321	0.6532	0.4819	0.6702	0.7523	0.7642	0.5401	0.8421	0	0.2207	0.6627	0.7011
Multinomial Naive Bayes	0.7357	0.3588	0.6811	0.6517	0.0756	0.6932	0.8187	0.7997	0	0.8039	0	0	0	0.1046
SVM	0.8706	0.736	0.8679	0.8294	0.7229	0.8415	0.8812	0.876	0.6838	0.9448	0.2222	0.4607	0.7372	0.8964
Logistic Regression	0.8446	0.6582	0.8439	0.8225	0.6812	0.8203	0.8523	0.8612	0.5891	0.9301	0	0.2621	0.6395	0.83
KNN one-vs-rest	0.7329	0.5426	0.7314	0.653	0.4706	0.6866	0.7701	0.7577	0.1979	0.8476	0	0.2289	0.6618	0.6944
Naive Bayes one-vs-rest	0.7102	0.3833	0.6825	0.5526	0.2472	0.5333	0.7539	0.7739	0	0.8932	0	0	0.0368	0.4252
Logistic Regression one-vs-rest	0.8507	0.6611	0.8445	0.8248	0.6730	0.8238	0.8541	0.862	0.5789	0.9320	0	0.2548	0.6383	0.8299
1D CNN	0.8553	0.6808	0.8523	0.8123	0.7138	0.8239	0.8682	0.8662	0.625	0.9451	0	0.2047	0.7863	0.8438
LSTM	0.8431	0.6231	0.8364	0.7718	0.6738	0.807	0.863	0.8605	0.3671	0.9491	0	0.0488	0.6945	0.8183
Supports	16702	16702	16702	1142	735	1585	2505	3550	133	3576	22	145	172	425

Table 5-3: Accuracy metrics from topic labeling on US Consumer Finance Complaint

As it can be inferred from Figure 5-2 and Table 5-3, the overall best performance is again achieved by the SVM model with unigrams and bigrams TF-IDF, followed by 1D CNN model embedding dimension size of 100 fitted from our own dataset, then, Logistic Regression with the same setting of feature engineering with the SVM model. The SVM model scores highest in both macro F-1 Score and accuracy. It shows 87.06% of accuracy and F-1 macro of 73.60%, whereas 1D CNN shows 85.53% accuracy and F-1 macro of 68.08%. 1D CNN outperformed LSTM and Logistic Regression model by a small margin.

The result is in accordance with Yao et al., (2018) where they mentioned that Logistics Regression with TF-IDF performs well on long text like 20NG. Our US Consumer Finance Complaint dataset has an average length of 190.64 words, whereas the 20NG dataset has an everage length of 221.26 words.

This emphasizes the robustness of the SVM model to text classification problems. The reason that deep learning models did not show the highest accuracy among all is, perhaps, the dataset is heavily unbalanced and the size of the data is not large enough. Besides, it is obvious that Multinomial Naïve Bayes performed poorly to small classes, especially in this topic labeling problem. KNN's performance is moderate among all models we tested.

For one-versus-rest models, Grid Search technique was used to find best parameters for each of the binary model. To classify the document, given that we have r classes, it starts from the model 1, if model 1 classifies as "False", it will proceed to use model 2 to classify, otherwise it will stop. It will proceed until model r if none of the prior models classify the response as "True". The model 1 to model r was ranked by the size, sorted in ascending order. This is to prioritize smaller classes. However, one-versus rest for conventional models did not show any significant difference than

multi-class conventional models. This is because the size of the minority classes is extremely small, therefore, those binary models failed to learn.

The selection of hyperparameters for the machine learning models for this study is shown in Table 5-2.

5.3 Feature engineering and selection of hyperparameters

Table 5-4: Hyperparameters used for machine learning models

Model	Hyperparameters for sentiment analysis problem	Hyperparameters for topic labeling problem
K-Nearest Neighbors	Remove stop words TF-IDF vectorizer Min df = 5 n-gram range = (1,2)	Remove stop words TF-IDF vectorizer Min df = 5 n-gram range = (1,2) k = 3
Multinomial naïve Bayes	Remove stop words TF-IDF vectorizer Min df = 5 n-gram range = (1,2) Additive smoothing parameter = 1	Remove stop words TF-IDF vectorizer Min df = 5 n-gram range = (1,2) Additive smoothing parameter = 1
Support Vector Machine	Remove stop words TF-IDF vectorizer Min df = 5 n-gram range = (1,2) Kernel= linear, regularization parameter C=1, penalty norm = L^1	Remove stop words TF-IDF vectorizer Min df = 5 n-gram range = (1,2) Kernel= linear, regularization parameter C=1, penalty norm = L^1
Logistic Regression	Regularization parameter C =1, penalty norm = L^1	Regularization parameter C =1, penalty norm = L^1
One-vs-rest K-Nearest Neighbors	Test not performed	(each of the 11 binary classifiers can have parameters fall in the ranges below) Stop words = [‘english’, None] n-gram = None TF-IDF vectorizer Min df list = [1,2,3] k list = [3,5,7]

One-vs-rest Binomial naïve Bayes	Test not performed	(each of the 11 binary classifiers can have parameters fall in the ranges below) Stop words = [‘english’, None] n-gram = None TF-IDF vectorizer Min df list = [1,2,3] Max df list = [0.995 , 0.999, 1.0] Additive smoothing parameter range = [0.5, 1.0, 2.0 , 5.0]
One-vs-rest Logistic Regression	Test not performed	(each of the 11 binary classifiers can have parameters fall in the ranges below) Stop words = [‘english’, None] n-gram = None TF-IDF vectorizer Min df list = [1,2,3]
1-D Convolutional Neural Network	Remove stop words & punctuations Max Number of words = 50,000 Max Sequence length = 256 Embedding Dimension = 100 Number of filters = 128 Kernel Size = 3 Size of max pooling windows = 2 Padding = ‘same’ Activation at the last layer = ‘softmax’ Optimizer = ‘rmsprop’ Loss function = ‘categorical cross-entropy’ Epochs = 10 Batch size = 128 Drop out rate = 0.5	Remove stop words & punctuations Max Number of words = 50,000 Max Sequence length = 256 Embedding Dimension = 100 Number of filters = 128 Kernel Size = 5 Size of max pooling windows = 5 Padding = ‘same’ Activation at the last layer = ‘softmax’ Optimizer = ‘rmsprop’ Loss function = ‘categorical cross-entropy’ Epochs = 4 Batch size = 128 Drop out rate = 0.5
Long short-term memory	Remove stop words & punctuations Max Number of words = 50,000 Max Sequence length = 250 Embedding Dimension = 100 Number of neurons in LSTM unit = 100	Remove stop words & punctuations Max Number of words = 50,000 Max Sequence length = 250 Embedding Dimension = 100 Number of neurons in LSTM unit = 100 Activation at the last layer = ‘softmax’ Optimizer = ‘adam’

	Activation at the last layer = 'softmax' Optimizer = 'adam' Loss function = 'categorical cross-entropy' Epochs = 5 Batch size = 64 Drop out rate = 0.2	Loss function = 'categorical cross-entropy' Epochs = 5 Batch size = 64 Drop out rate = 0.2

For rules and thresholds set for VADER model for this study is shown in Table 5-5.

Table 5-5: Rules and thresholds used for VADER model for sentiment classification

Model	Rules and Thresholds

VADER	# snippet of the code for 5-class sentiment prediction
	def classify_review_star(row):
	  if row['compound_score']>=0.05:
	  if row['compound_score']>0.185:
	  return 5
	  else:
	  return 4
	  elif (row['compound_score']> -0.05) and(row['compound_score']< 0.05):
	  return 3
	  elif row['compound_score']<= -0.05:
	  if row['negative_score']>0.108:
	  return 1
	  else:
	  return 2

Talking about selecting options in feature engineering steps and hyperparameters for training, in this study, considering a pair of words as the same token (2-grams) has proven beneficial in uncovering the underlying semantic relationships between the words as it resulted in better classification performances. Count vectorizer method did not show great performance, therefore, it was not included here. This justifies the popularity of using conventional classifiers in high dimensional setup. Multinomial naïve Bayes classifier was assigned with a smoothing parameter;

a value 1 is added to the word count while calculating the probability to avoid the zero probability in case that the word has never appeared before in that particular class. Talking about logistic regression and support vector machine, the regularization parameter used is 1 and the penalty norm used is 'L-1' norm. 'L-1' norm is used as it aids in reducing the dimensionality of the matrix by providing a sparse solution with very few non-zero coefficients (Pedregosa et al., 2011).

Apart from apart from the baseline 1D CNN model, we also perform the static 1-D CNN with GloVe as well as Google Word2vec embedding which is demonstrated by Kim, Y. (2014). Convolutional and pooling layers are split into 3 branches which has filter window size of 3,4,5, but the convolutional and pooling layers are split into 3 branches. The result from 3 branches are later concatenated into a single vector at the flatten layer. This is design has similar idea to 3,4,5 grams in TF-IDF method. However, for our dataset, this design did not gives higher accuracy than the baseline 1D CNN that did not split the branches but play with number convolutional and pooling layers. Therefore, the result from 3-branches 1-D CNN is not included here.

To fine-tune the deep learning models, for both 1D CNN and LSTM models, we set the maximum number of words (tokens) to 50,000. After removing the stop words and punctiations, a total of 199,055 tokens in Amazon data and 10,0081 tokens in US Consumer complaint data are found. Only 50,000 words will be considered in the deep learning models. For 1D CNN, we constructed convolutional layers that each of them is followed by a pooling layer with "max pooling" method. After that, we convert the output into a 1-dimension feature vector by using the flattening layer, and that single long vector will be used by the dense layer with 'softmax' activation to perform multi-class classification. For LSTM's case, we specify the maximum length of the sentence and padding, number memory cells in each of the LSTM units spatial dropout rate, recurrent dropout rate, and the softmax classifier as the output layer.

Apart from selecting the model architecture and hyperparameters, for each setting, we also need to find the optimum point to prevent overfitting.

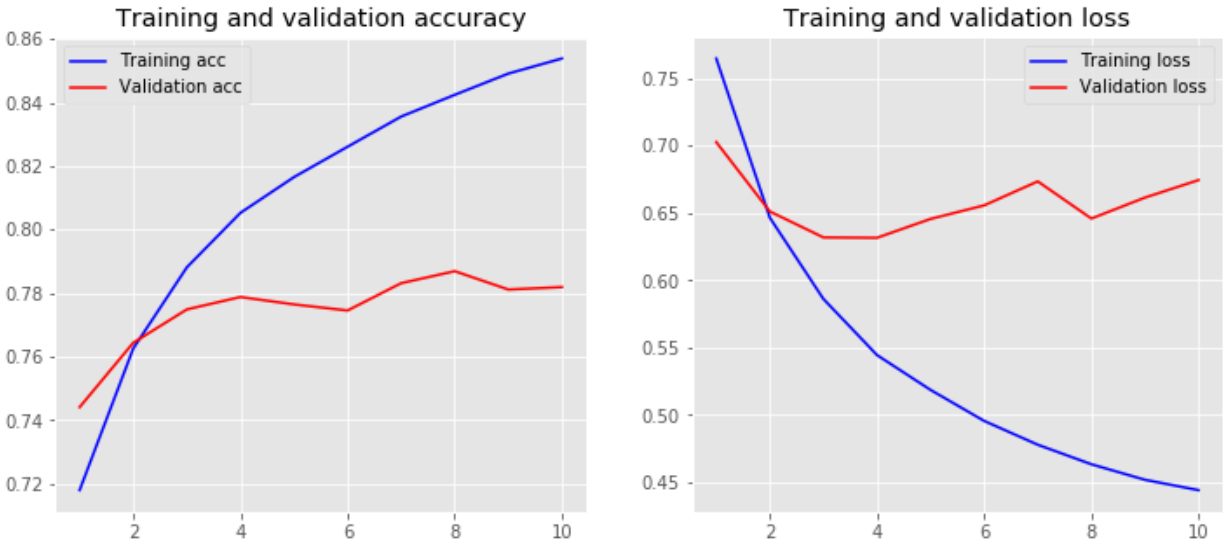


Figure 5-3: Training and validation graphs for deep learning models (left) training accuracy and validation accuracy of each epoch (1-10); (right) training loss and validation loss of each epoch (1-10)

Figure 5-3 is an example taken from one of the settings of 1D Convolutional Neural Network models from our study. Originally, the number of epochs was set to be 10 which is higher than the optimum point. After finishing 10 epochs of training, we checked the graphs of training and validation in order to find the optimum iterations of training. Figure 5-3 (right) shows that the loss of validation data starts rising again after the 4th epoch. At the same point, the validation accuracy tends to be stable but the training accuracy tends to go higher. The point at which the validation accuracy and train accuracy begins to diverge indicates that the model has been trained and starts overfitting. In this case, the 4th epoch tends to be a good point to stop the training since the model seems to generalized to unseen data and not overfitting. After this analysis, we re-fit the model again and stop at that such optimum number of epochs we have

determined. Keras also offer “early stopping” function to prevent over training the neural network models.

Chapter 6
Conclusions and Future work

In this study, we focus on automated text classification systems for online reviews. This study deals with two problems which are predicting the sentiment and classifying the topic of the online reviews. The purpose of this study is to help business firms in providing a better understanding of customer's sentiment towards the products as well as identifying the customer's problem so that timely remedial actions can be made. Conventional machine learning models, namely, K-Nearest Neighbors (KNN), Multinomial naïve Bayes, Logistic regression, Support Vector Machine (SVM), as well as deep learning models and one rule-based model are used for both sentiment classification and topic labeling tasks. The "one-versus-rest" strategy was also tested for conventional models to evaluate their performance against the multi-class classifiers.

Additionally, one rule-based model named VADER was experimented in 5-class sentiment classification, which to the best of our knowledge, has never been attempted before.

The key findings of this study are discussed below:

- Support Vector Machine (SVM) performed consistently well. This study emphasizes the robustness of SVM in text classification. Interestingly, SVM slightly outperformed Long Short-Term Memory (LSTM) model and 1D Convolutional Neural Network (1D CNN) in both sentiment classification task and topic modeling task. Moreover, SVM is comparatively faster to train. SVM has the highest overall accuracy and macro F-1 score. The reason that deep learning models did not perform best is due to the nature and class distribution of the datasets in our study.
- VADER model only gives high F-1 score in the extremely positive sentiment class. It yields low overall accuracy in classifying 5 levels of sentiment. The reason is that the

compound, positive, and negative scores predicted from VADER for 5 classes are not mutually independent. This is as expected because VADER is specifically created for only 3-class sentiment classification (negative, neutral and positive) and also this model considers only about 7,500 words. In this study, we manually set a simple rule-based model to classify the class on top of the positive, negative, and compound scores from VADER.

- The datasets used in two problems are imbalanced, especially in topic labeling problem where the smallest class has only 105 observations (less than 1% compared to the biggest class). The SVM model also tends to learn better in minority classes compared to other models in this study. This results in a comparatively high macro F-1 score in SVM's case.
- The one-vs-rest strategy did not show a significant improvement in accuracy against multi-class classifiers of conventional models in our study of topic labeling since the minority classes are too small for machine learning models to learn well.

This research provides ample scope for potential future work and can be moved forward in multiple directions. In this study, the deep learning models were used in EC2 Medium dedicated hardware tier on Amazon AWS. In the future, more powerful GPUs can be used to reduce the time for hyperparameter tuning in the training process. For deep learning methods, part-of-speech tag (POS tag) can be added to make the input matrices carry more information. Another idea to improve is to merge the smallest class into a class that has high similarity before performing the test to mitigate the class-imbalance. We can try to artificially balance the training set by using undersampling or oversampling techniques such as RUS or SMOTE. Lastly, it can also be checked how better VADER will perform compared to other algorithms if we convert the

targets of Amazon Fine Food Review from 5-star (5 levels) to be only 3-level. Lastly, we examine whether using classifiers such as Multi-Layer Perceptron (MLP) on top of the scores predicted from VADER will make a significant improvement.

Appendix
Python code

Analysis of vectorized features

Conv1D model

```
# Download stopwords from NLTK, type this command in the console
python3 nltk
import nltk
nltk.download('stopwords')

# Import all libraries and packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
from keras.callbacks import EarlyStopping
from keras.layers import Dropout
import re
from nltk.corpus import stopwords
from nltk import word_tokenize
STOPWORDS = set(stopwords.words('english'))
from bs4 import BeautifulSoup
import plotly.graph_objs as go
import plotly.plotly as py
import cufflinks
from IPython.core.interactiveshell import InteractiveShell
import plotly.figure_factory as ff
InteractiveShell.ast_node_interactivity = 'all'
from plotly.offline import iplot
cufflinks.go_offline()
cufflinks.set_config_file(world_readable=True, theme='pearl')

import tensorflow as tf
from keras.backend.tensorflow_backend import set_session
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
config.log_device_placement = True
sess = tf.Session(config=config)
set_session(sess)
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential, Model
```

```
from keras.layers.convolutional import Conv1D, MaxPooling1D
from keras.layers import Dense, Embedding, LSTM, CuDNNLSTM, Dropout, Flatt
en, Input
# setting max number of words (tokens) to be used in the model
num_words = 50000
```

- **Conv1D : For Sentiment Analysis (Amazon Fine Food)**

```
# load the data, show the overview of the data , and perform data preproce
-ssing, by removing punctuations and stopwords, and perform tokenization,
and specify the word embedding dimension
df = pd.read_csv('Reviews_2.csv')
df.Score.value_counts()
df['Score'].value_counts().sort_values(ascending=False).iplot(kind='bar',
yTitle='Number of Reviews', title='Number reviews in each rating')


def print_plot(index):
    example = df[df.index == index][['Text', 'Score']].values[0]
    if len(example) > 0:
        print(example[0])
        print('Score:', example[1])


df = df.reset_index(drop=True)
REPLACE_BY_SPACE_RE = re.compile('[/(){}\[\]\|@,;]')
BAD_SYMBOLS_RE = re.compile('[^0-9a-z #+_]')
STOPWORDS = set(stopwords.words('english'))

def clean_text(text):
    """
        text: a string

        return: modified initial string
    """
    text = text.lower() # lowercase text
    text = REPLACE_BY_SPACE_RE.sub(' ', text)

# replace REPLACE_BY_SPACE_RE symbols by space in text. substitute the mat
ched string in REPLACE_BY_SPACE_RE with space.
    text = BAD_SYMBOLS_RE.sub('', text) # remove symbols which are in BAD_
SYMBOLS_RE from text. substitute the matched string in BAD_SYMBOLS_RE with
nothing.
    text = text.replace('x', '')
    text = ' '.join(word for word in text.split() if word not in STOPWORDS
) # remove stopwors from text
    return text
df['Text'] = df['Text'].apply(clean_text)
df['Text'] = df['Text'].str.replace('\d+', '')
```

```
# Maximum number of words to be used (most frequent)
MAX_NB_WORDS = 50000
# Maximum number of words in each review or complaint.
MAX_SEQUENCE_LENGTH = 150
# Specify the dimension for word embedding
EMBEDDING_DIM = 50

tokenizer = Tokenizer(num_words=MAX_NB_WORDS, filters='!"#$%&()*+,-./:;<=>
?@[\]^_`{|}~', lower=True)
tokenizer.fit_on_texts(df['Text'].values)
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))

X = tokenizer.texts_to_sequences(df['Text'].values)
X = pad_sequences(X, maxlen=MAX_SEQUENCE_LENGTH)
print('Shape of data tensor:', X.shape)

Y = pd.get_dummies(df['Score']).values
print('Shape of label tensor:', Y.shape)

# Split the dataset into train set and test set
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.10,
random_state = 42)
print(X_train.shape,Y_train.shape)
print(X_test.shape,Y_test.shape)
```

- **Conv1D : Start building the CNN model**

```
#maximum length including padding
max_len = 256
#size of word embedding
embedding_size = 50
#batch size that will be used used in one iteration of training
batch_size = 128

pad  =  'post' #'pre'
#Convert our lists to equal length sequences
X_train_pad = pad_sequences(X_train, maxlen=max_len, padding=pad, truncati
ng=pad)
X_test_pad = pad_sequences(X_test, maxlen=max_len, padding=pad, truncating
=pad)

model0 = Sequential()
model0.add(Embedding(input_dim=num_words,
                     output_dim=embedding_size,
                     input_length=max_len,
                     name='layer_embedding'))

# Specify the number of filters ,kernel size, padding method, activation f
unction for the convolutional layer,
model0.add(Conv1D(filters=128, kernel_size=5, padding='same', activation='
relu'))
# followed by specifying the pooing window size of the pooling layer with
max pooling method
```

```
model0.add(MaxPooling1D(pool_size=5))
model0.add(Conv1D(filters=128, kernel_size=5, padding='same', activation='
relu'))
model0.add(MaxPooling1D(pool_size=5))
model0.add(Conv1D(filters=128, kernel_size=5, padding='same', activation='
relu'))
model0.add(MaxPooling1D(pool_size=5))
# Specify the drop out rate
model0.add(Dropout(0.2))
# Add the flatten layer
model0.add(Flatten())
# Add the fully connected layer
model0.add(Dense(250, activation='relu'))
# Add the final classification layer using softmax activation
model0.add(Dense(5, activation='softmax'))
# Compile the model
model0.add(Dense(5, activation='softmax'))
# Show model summary
model0.summary()
```

- **Conv1D : Train the model and show only the results from training-validation but not plotting the training-validation graphs**

```
%% time
# Specify the train-validation ratio, number of epochs of training, and th
en train-validation ratio
model0.fit(X_train_pad, Y_train,
                    epochs=10,
                    validation_split=0.2222,
                    batch_size=batch_size)
```

- **Conv1D : Train the model and plot the training-validation graphs for CNN models to show training accuracy vs validation accuracy for each epoch; training loss vs validation loss for each epoch (optional)**

```
import matplotlib.pyplot as plt
plt.style.use('ggplot')

def plot_history(history):
    acc = history.history['acc']
    val_acc = history.history['val_acc']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    x = range(1, len(acc) + 1)

    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(x, acc, 'b', label='Training acc')
    plt.plot(x, val_acc, 'r', label='Validation acc')
```

```
    plt.title('Training and validation accuracy')
    plt.legend()
    plt.subplot(1, 2, 2)
    plt.plot(x, loss, 'b', label='Training loss')
    plt.plot(x, val_loss, 'r', label='Validation loss')
    plt.title('Training and validation loss')
    plt.legend()

history = model0.fit(X_train_pad, Y_train,
                     epochs=10,
                     verbose=False,
                     validation_split=0.2222,
                     batch_size=batch_size)
loss, accuracy = model0.evaluate(X_train_pad, Y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))
loss, accuracy = model0.evaluate(X_test_pad, Y_test, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy))
plot_history(history)
```

- **Conv1D : Testing and evaluation**

```
# Evaluate the model by performing classification on test dataset
eval_ = model0.evaluate(X_test_pad, Y_test)
# Show the accuracy
print("Loss: {0:.5}".format(eval_[0]))
print("Accuracy: {0:.2%}".format(eval_[1]))

# Show accuracy metrics such as overall accuracy, f-1 macro, f-1 weighted,
f-1 score for each particular class, ROC , AUC, confusion matrix
from sklearn.datasets import make_circles
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix

def decode(datum):
    return np.argmax(datum)
Y_test = np.array([decode(i) for i in Y_test])
yhat_probs = model0.predict(X_test_pad, verbose=0)
# predict crisp classes for test set
yhat_classes = model0.predict_classes(X_test_pad, verbose=0)
# Show the overall accuracy: (tp + tn) / (p + n)
accuracy = accuracy_score(Y_test, yhat_classes)
print('Accuracy: %f' % accuracy)
# Show f-1 macro
f1_score(Y_test, yhat_classes, average='macro')
# Show f-1 score for each particular class
f1_score(Y_test, yhat_classes, average=None)
# Show f-1 weighted
f1_score(Y_test, yhat_classes, average='weighted')
# Show confusion matrix
```

```
matrix = confusion_matrix(Y_test, yhat_classes)
print(matrix)
```

- **3,4,5 grams Conv1D: Building the model**

```
# credit : https://machinelearningmastery.com/develop-n-gram-multichannel-convolutional-neural-network-sentiment-analysis/

# define the model
from keras.utils.vis_utils import plot_model
def define_model(length, vocab_size):
    # channel 1
    inputs1 = Input(shape=(length,))
    embedding1 = Embedding(vocab_size, 300, weights=[wv_matrix], trainable=False)(inputs1)
    conv1 = Conv1D(filters=100, kernel_size=3, activation='relu')(embedding1)
    drop1 = Dropout(0.5)(conv1)
    pool1 = MaxPooling1D(pool_size=2)(drop1)
    flat1 = Flatten()(pool1)
    # channel 2
    inputs2 = Input(shape=(length,))
    embedding2 = Embedding(vocab_size, 300, weights=[wv_matrix], trainable=False)(inputs2)
    conv2 = Conv1D(filters=100, kernel_size=4, activation='relu')(embedding2)
    drop2 = Dropout(0.5)(conv2)
    pool2 = MaxPooling1D(pool_size=2)(drop2)
    flat2 = Flatten()(pool2)
    # channel 3
    inputs3 = Input(shape=(length,))
    embedding3 = Embedding(vocab_size, 300, weights=[wv_matrix], trainable=False)(inputs3)
    conv3 = Conv1D(filters=100, kernel_size=5, activation='relu')(embedding3)
    drop3 = Dropout(0.5)(conv3)
    pool3 = MaxPooling1D(pool_size=2)(drop3)
    flat3 = Flatten()(pool3)
    # merge
    merged = concatenate([flat1, flat2, flat3])
    # interpretation
    dense1 = Dense(64, activation='relu')(merged)
    outputs = Dense(20, activation='softmax')(dense1)
    model = Model(inputs=[inputs1, inputs2, inputs3], outputs=outputs)
    # compile
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    # summarize
    print(model.summary())
    # plot_model(model, show_shapes=True, to_file='multichannel.png')
```

```
    return model

model0 = define_model(max_len, num_words)
```

- **3,4,5 grams Conv1D: Training, Testing and evaluation**

```
# train model

model0.fit([X_train_pad,X_train_pad,X_train_pad], np.array(Y_train), epoch
s=6, batch_size=50, validation_split=0.10)

# evaluate model on test dataset dataset
loss, acc = model0.evaluate([X_test_pad,X_test_pad,X_test_pad], np.array(Y
_test), verbose=0)
print('Test Accuracy: %f' % (acc*100))

# show the accuracy metrics

from sklearn.datasets import make_circles
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report


def decode(datum):
    return np.argmax(datum)

Y_test = np.array([decode(i) for i in Y_test])
y_true = Y_test
y_pred = model0.predict([X_test_pad,X_test_pad,X_test_pad])
val_preds = np.argmax(y_pred, axis=-1)
print(classification_report(y_true, val_preds,digits=4))
# confusion matrix
matrix = confusion_matrix(y_true, val_preds)
print(matrix)
```

LSTM model

```
# Download stopwords from NLTK, type this command in the console
python3 nltk
import nltk
nltk.download('stopwords')

# Import all libraries and packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
import seaborn as sns
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
from keras.callbacks import EarlyStopping
from keras.layers import Dropout
import re
from nltk.corpus import stopwords
from nltk import word_tokenize
#STOPWORDS = set(stopwords.words('english'))
from bs4 import BeautifulSoup
import plotly.graph_objs as go
import plotly.plotly as py
import cufflinks
from IPython.core.interactiveshell import InteractiveShell
import plotly.figure_factory as ff
InteractiveShell.ast_node_interactivity = 'all'
from plotly.offline import iplot
cufflinks.go_offline()
cufflinks.set_config_file(world_readable=True, theme='pearl')
```

- **LSTM : For Sentiment Analysis (Amazon Fine Food)**

```
# load the data, show the overview of the data , and perform data preproce
-ssing, by removing punctuations and stopwords, and perform tokenization,
and specify the word embedding dimension

df = pd.read_csv('Reviews_2.csv')
df = df[['Text','Score']]
df.Score.value_counts()
# Show bar plot of number of reviews in each rating
df['Score'].value_counts().sort_values(ascending=False).iplot(kind='bar',
yTitle='Number of Reviews',title='Number reviews in each rating')
STOPWORDS = set(stopwords.words('english'))
df = df.reset_index(drop=True)
REPLACE_BY_SPACE_RE = re.compile('[/(){}\[\]\|@,;]')
BAD_SYMBOLS_RE = re.compile('[^0-9a-z #+_]')
STOPWORDS = set(stopwords.words('english'))

def clean_text(text):
    """
        text: a string
        return: modified initial string
    """
    text = text.lower() # lowercase text
    text = REPLACE_BY_SPACE_RE.sub(' ', text)
# replace REPLACE_BY_SPACE_RE symbols by space in text. substitute the mat
ched string in REPLACE_BY_SPACE_RE with space.
    text = BAD_SYMBOLS_RE.sub('', text)
# remove symbols which are in BAD_SYMBOLS_RE from text. substitute the mat
ched string in BAD_SYMBOLS_RE with nothing.
```

```
    text = text.replace('x', '')
    text = ' '.join(word for word in text.split() if word not in STOPWORDS
) # remove stopwors from text
    return text
df['Text'] = df['Text'].apply(clean_text)
df['Text'] = df['Text'].str.replace('\d+', '')
# Specify the maximum number of words to be used (most frequent)
MAX_NB_WORDS = 50000
# Specify the maximum number of words in each review or complaint.
MAX_SEQUENCE_LENGTH = 250
# Size of word embedding
EMBEDDING_DIM = 100

tokenizer = Tokenizer(num_words=MAX_NB_WORDS, filters='!"#$%&()*+,-./:;<=>
?@[\]^_`{|}~', lower=True)
tokenizer.fit_on_texts(df['Text'].values)
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
X = tokenizer.texts_to_sequences(df['Text'].values)
X = pad_sequences(X, maxlen=MAX_SEQUENCE_LENGTH)
print('Shape of data tensor:', X.shape)
Y = pd.get_dummies(df['Score']).values
print('Shape of label tensor:', Y.shape)
```

- **LSTM : Start building the LSTM model**

```
model = Sequential()
# Specify the shape of the input
model.add(Embedding(MAX_NB_WORDS, EMBEDDING_DIM, input_length=X.shape[1]))
# SpatialDrpout1D performs variational dropout
model.add(SpatialDropout1D(0.2))
# Add the LSTMs and specify number memory units for LSTMs
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
# Add the final classification layer using softmax activation
model.add(Dense(5, activation='softmax'))
# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=[
'accuracy'])
# Show model summary
(model.summary())
```

- **LSTM : Train the model**

```
#Specify number of epochs and batch size
epochs = 5
batch_size = 64


# Specify the train-validation ratio, number of epochs of training, and th
en train-validation ratio
history = model.fit(X_train, Y_train, epochs=epochs, batch_size=batch_size
,validation_split=0.1,callbacks=[EarlyStopping(monitor='val_loss', patienc
e=3, min_delta=0.0001)])
```

- **LSTM: Plot the training-validation graphs for LSTM model to show training accuracy vs validation accuracy for each epoch; training loss vs validation loss for each epoch (optional)**

```
plt.title('Loss')
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()

plt.title('Accuracy')
plt.plot(history.history['acc'], label='train')
plt.plot(history.history['val_acc'], label='test')
plt.legend()
plt.show()
```

- **LSTM : Perform testing and evaluation and show accuracy metrics**

```
# Evaluate the model by performing classification on test dataset
accr = model.evaluate(X_test,Y_test)
print('Test set\n  Loss: {:0.3f}\n  Accuracy: {:0.3f}'.format(accr[0],accr
[1]))
# Show accuracy metrics such as overall accuracy, f-1 macro, f-1 weighted,
f-1 score for each particular class, ROC , AUC, confusion matrix
from sklearn.datasets import make_circles
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix

def decode(datum):
    return np.argmax(datum)
Y_test = np.array([decode(i) for i in Y_test])
yhat_probs = model0.predict(X_test_pad, verbose=0)
# predict crisp classes for test set
yhat_classes = model0.predict_classes(X_test_pad, verbose=0)
# Show the overall accuracy: (tp + tn) / (p + n)
accuracy = accuracy_score(Y_test, yhat_classes)
print('Accuracy: %f' % accuracy)
# Show f-1 macro
f1_score(Y_test, yhat_classes, average='macro')
# Show f-1 score for each particular class
f1_score(Y_test, yhat_classes, average=None)
# Show f-1 weighted
f1_score(Y_test, yhat_classes, average='weighted')
# Show confusion matrix
matrix = confusion_matrix(Y_test, yhat_classes)
```

```
print(matrix)
```

GloVe Embedding for deep learning models

```
# Glove Embedding file of dimension 50, 100, 200, 300 can be downloaded fr
om http://nlp.stanford.edu/data/glove.6B.zip

# Load 50 dimension GloVe embedding vectors
embeddings_index = dict()
f = open('./glove.6B.50d.txt')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

# Assign a 50 dimension GloVe embedding vector to each word in the dataset
embedding_matrix = np.zeros((MAX_NB_WORDS, 50))
for word, index in tokenizer.word_index.items():
    if index > MAX_NB_WORDS - 1:
        break
    else:
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[index] = embedding_vector
```

Gloogle News Embedding for deep learning models

```
# Google News Embedding file of dimension 300 can be downloaded from https
://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit


# Load 300 dimension Google embedding vectors


from gensim.models import KeyedVectors
filename = '../pretrained_embeddings/Google_w2v/GoogleNews-vectors-negativ
e300.bin'
model = KeyedVectors.load_word2vec_format(filename, binary=True)

# Assign a 300 dimension Google News embedding vector to each word in the
dataset
MAX_NB_WORDS = len(word_vectors.vocab)
WV_DIM = 300
nb_words = min(MAX_NB_WORDS, len(word_vectors.vocab))
# we initialize the matrix with random numbers
wv_matrix = (np.random.rand(nb_words, WV_DIM) - 0.5) / 5.0
for word, i in word_index.items():
    if i >= MAX_NB_WORDS:
        continue
    try:
        embedding_vector = word_vectors[word]
```

```
        # words not found in embedding index will be all-zeros.
        wv_matrix[i] = embedding_vector
    except:
        pass
```

Conventional models (Support Vector Machine, Naïve Bayes, Logistic Regression, K-Nearest Neighbors)

```
# Load the data, show the overview of the data , and perform data preproce
-ssing, by removing punctuations and stopwords, and perform tokenization
import pandas as pd
df = pd.read_csv('Reviews_2.csv')
df = df[['Text','Score']]
df.head()
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(8,6))
df.groupby('Score').Text.count().plot.bar(ylim=0)
plt.show()

# Import TF-IDF vectorizer(or Count vectorizer), specify the parameters an
d perform vectorization
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(sublinear_tf=True, min_df=5, norm='l2', encoding='
latin-1', ngram_range=(1, 2), stop_words='english')
features = tfidf.fit_transform(df.Text)
labels = df.category_id
features.shape
```

- **Conventional models : Train the model, validate the model and show the result from k-fold validation**

```
# Import Conventional models and perform k-fold validation to compare the
accuracy
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score

# List all the models we want to train and validate using k-fold
models = [RandomForestClassifier(n_estimators=200, max_depth=3, random_sta
te=0),LinearSVC(), MultinomialNB(), LogisticRegression(random_state=0)]
CV = 5
cv_df = pd.DataFrame(index=range(CV * len(models)))
entries = []
for model in models:
  model_name = model.__class__.__name__
```

```
  accuracies = cross_val_score(model, features, labels, scoring='accuracy'
, cv=CV)
  for fold_idx, accuracy in enumerate(accuracies):
    entries.append((model_name, fold_idx, accuracy))
cv_df = pd.DataFrame(entries, columns=['model_name', 'fold_idx', 'accuracy
'])
import seaborn as sns
sns.boxplot(x='model_name', y='accuracy', data=cv_df)
sns.stripplot(x='model_name', y='accuracy', data=cv_df,
              size=8, jitter=True, edgecolor="gray", linewidth=2)
plt.show()

# Show the averaged accuracy from k-fold validation for each of the conven
tional models
cv_df.groupby('model_name').accuracy.mean()
```

Conventional models : Re-fit the model (fit the model again with the training set), then perform testing and evaluation on test set

```
#1. Support Vector Machine model
model = LinearSVC()
X_train, X_test, y_train, y_test, indices_train, indices_test = train_test
_split(features, labels, df.index, test_size=0.10, random_state=0)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

#2.  Multinomial Naïve Bayes model
model = MultinomialNB()
X_train, X_test, y_train, y_test, indices_train, indices_test = train_test
_split(features, labels, df.index, test_size=0.10, random_state=0)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

#3.  Logistic Regression model

#3.1  For Multinomial Logistic Regression model, specify the parameter mul
ti_class = "ovr" and solver = "lbfgs"

#3.2  For One-vs-rest strategy binary Logistic Regression model, specify t
he parameter multi_class = "ovr"
model = LogisticRegression(random_state=0)
X_train, X_test, y_train, y_test, indices_train, indices_test = train_test
_split(features, labels, df.index, test_size=0.10, random_state=0)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

#4.  K-Nearest Neighbors model
model = KNeighborsClassifier(n_neighbors=3)
X_train, X_test, y_train, y_test, indices_train, indices_test = train_test
_split(features, labels, df.index, test_size=0.10, random_state=0)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

Conventional models : Show accuracy metrics

```
# Show confusion matrix
conf_mat = confusion_matrix(y_test, y_pred)
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(conf_mat, annot=True, fmt='d',
            xticklabels=category_id_df['Score'].values, yticklabels=catego
ry_id_df['Score'].values)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()

# Show accuracy metrics, which are precision, recall and f-1 score for eac
h class, f-1 macro, f-1 weighted overall accuracy
from sklearn import metrics
print(metrics.classification_report(y_test, y_pred, digits=4, target_names
=df['Score'].apply(str).unique()))
```

VADER model

- **VADER: Predict the compound, positive, negative scores**

```
# Import the library and load the data
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
import pandas as pd
df = pd.read_csv("Reviews_2.csv")
df = df[['Text','Score']]

df['compound_score'] = None
df['positive_score'] = None
df['negative_score'] = None

# Let VADER model predict the compound, positive and negative scores for e
ach document (customer review from Amazon dataset)
for i in range(0,len(df)):
    vs = analyzer.polarity_scores(df['Text'][i])
    df['compound_score'][i] = vs['compound']
    df['positive_score'][i] = vs['pos']
    df['negative_score'][i] = vs['neg']
```

- **VADER: Explore to see the stats and distribution of the scores predicted by VADER model**

```
# Show stats (e.g. min, max, count, mean median) of the predicted scores (
compound, positive, negative) for each review-star rating
df.groupby('Score').describe()

# Show the normalized distribution of compound score
x1 = df[df['Score']==1]['compound_score']
```

```
x2 = df[df['Score']==2]['compound_score']
x3 = df[df['Score']==3]['compound_score']
x4 = df[df['Score']==4]['compound_score']
x5 = df[df['Score']==5]['compound_score']
kwargs = dict(histtype='step', alpha=1, normed=True, bins=80)
plt.hist(x1, **kwargs)
plt.hist(x2, **kwargs)
plt.hist(x3, **kwargs)
plt.hist(x4, **kwargs)
plt.hist(x5, **kwargs);

# Show the normalized distribution of positive score
x1 = df[df['Score']==1]['positive_score']
x2 = df[df['Score']==2]['positive_score']
x3 = df[df['Score']==3]['positive_score']
x4 = df[df['Score']==4]['positive_score']
x5 = df[df['Score']==5]['positive_score']
kwargs = dict(histtype='step', alpha=1, normed=True, bins=80)
plt.hist(x1, **kwargs)
plt.hist(x2, **kwargs)
plt.hist(x3, **kwargs)
plt.hist(x4, **kwargs)
plt.hist(x5, **kwargs);

# Show the normalized of negative score
x1 = df[df['Score']==1]['negative_score']
x2 = df[df['Score']==2]['negative_score']
x3 = df[df['Score']==3]['negative_score']
x4 = df[df['Score']==4]['negative_score']
x5 = df[df['Score']==5]['negative_score']
kwargs = dict(histtype='step', alpha=1, normed=True, bins=80)
plt.hist(x1, **kwargs)
plt.hist(x2, **kwargs)
plt.hist(x3, **kwargs)
plt.hist(x4, **kwargs)
plt.hist(x5, **kwargs);
```

- **VADER: Set the rules and thresholds for 5-classes classification, and then perform classification**

```
# Set the rules and thresholds for classification in this function
def classify_review_star(row):
    if row['compound_score']>=0.05:
        if row['compound_score']>0.185:
            return 5
        else:
            return 4
    elif (row['compound_score']> -0.05) and (row['compound_score']< 0.05):
        return 3
    elif row['compound_score']<= -0.05:
        if row['negative_score']>0.108:
            return 1
```

```
        else:
            return 2
# Perform 5-classes classification
df['calculate_Star'] = df.apply(calculate_Star,axis=1)
```

- **VADER: Show accuracy metrics**

```
from sklearn.datasets import make_circles
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
Y_test = df['Score']
yhat_classes = df['calculate_Star']
# Show the overall accuracy: (tp + tn) / (p + n)
accuracy = accuracy_score(Y_test, yhat_classes)
print('Accuracy: %f' % accuracy)
# Show f-1 macro
f1_score(Y_test, yhat_classes, average='macro')
# Show f-1 score for each particular class
f1_score(Y_test, yhat_classes, average=None)
# Show f-1 weighted
f1_score(Y_test, yhat_classes, average='weighted')
# Show confusion matrix
matrix = confusion_matrix(Y_test, yhat_classes)
print(matrix)
```

One-vs-rest Logistic Regression

Please refer to section “Conventional Models” in this chapter, see model 3.2

One-vs-rest models (Naïve Bayes, K-Nearest Neighbors) for Topic Labeling problem

```
from matplotlib import pyplot as plt
import numpy as np
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn import svm
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.multiclass import OneVsRestClassifier
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn import svm
```

```
import seaborn as sns
import pandas as pd
from sklearn.preprocessing import MultiLabelBinarizer
```

- **One-vs-rest models: K-Nearest Neighbors**

```
def KNN_topic_classification(review_data, feature):
    X_train, X_test, y_train, y_test = train_test_split(review_data, featu
re , test_size=0.20, random_state=0)
    text_clf = Pipeline([('tfidf', TfidfVectorizer()), ('clf', KNeighborsC
lassifier())])

    k_range = list(range(1,10))
    parameters = {'tfidf__min_df':[1,2,3],
                  'tfidf__stop_words':[None,"english"],
                  'clf__n_neighbors': k_range }
    metric =  "f1_macro"

    gs_clf = GridSearchCV(text_clf, param_grid=parameters, scoring=metric,
cv=5)
    gs_clf = gs_clf.fit(X_train, y_train)

    for param_name in gs_clf.best_params_:
        print("{} : {}".format(param_name , gs_clf.best_params_[param_name
]))
    print("best f1 score:", gs_clf.best_score_)

    clf_k = gs_clf.best_params_["clf__n_neighbors"]
    tfidf_min_df = gs_clf.best_params_["tfidf__min_df"]
    tfidf_stop_words = gs_clf.best_params_["tfidf__stop_words"]

    classifier = Pipeline([
        ('tfidf', TfidfVectorizer(stop_words=tfidf_stop_words,\
                                  min_df=tfidf_min_df)),
        ('clf', KNeighborsClassifier(n_neighbors= clf_k))])

    clf = classifier.fit(X_train,  y_train)

    labels=sorted(np.unique(feature))
    labels = list(map(str, labels))
    predicted = classifier.predict(X_test)
    print(classification_report(y_test, predicted))
    return predicted

categ_data = df
classes = categ_data.Product.values
classes = [i.split(",") for i in classes]

mlb = MultiLabelBinarizer()
Y=mlb.fit_transform(classes)
classes_label = mlb.classes_
```

- **One-vs-rest models: Naïve Bayes**

```
def NB_topic_classification(review_data, feature):
    X_train, X_test, y_train, y_test = train_test_split(review_data, featu
re , test_size=0.25, random_state=0)
    text_clf = Pipeline([('tfidf', TfidfVectorizer()), ('clf', Multinomial
NB())])
    parameters = {'tfidf__min_df':[1, 2, 3],
                  'tfidf__max_df': [0.995 , 0.999, 1.0],
                  'tfidf__stop_words':[None,"english"],
                  'clf__alpha': [0.5, 1.0, 2.0 , 5.0]}
    metric =  "f1_macro"
    gs_clf = GridSearchCV(text_clf, param_grid=parameters, scoring=metric,
cv=5)
    gs_clf = gs_clf.fit(X_train, y_train)

    for param_name in gs_clf.best_params_:
        print("{} : {}".format(param_name , gs_clf.best_params_[param_name
]))
    print("best f1 score:", gs_clf.best_score_)

    clf_alpha = gs_clf.best_params_["clf__alpha"]
    tfidf_min_df = gs_clf.best_params_["tfidf__min_df"]
    tfidf_max_df = gs_clf.best_params_["tfidf__max_df"]
    tfidf_stop_words = gs_clf.best_params_["tfidf__stop_words"]

    classifier = Pipeline([
        ('tfidf', TfidfVectorizer(stop_words=tfidf_stop_words,\
                                  min_df=tfidf_min_df)),
        ('clf',  MultinomialNB(alpha = clf_alpha ))])

    clf = classifier.fit(X_train,  y_train)

    labels=sorted(np.unique(feature))
    labels = list(map(str, labels))

    predicted = classifier.predict(X_test)

    print(classification_report(y_test, predicted))

    return predicted

categ_data = df
classes = categ_data.Product.values
classes = [i.split(",") for i in classes]
mlb = MultiLabelBinarizer()
Y=mlb.fit_transform(classes)
classes_label = mlb.classes_
```

- **One-vs-rest models: perform Grid search cross validation (GridSearchCV) with number of fold = 5 to find the best binary model to predict each particular class;**

and perform prediction; and show the best parameters (example shown below is for Naïve Bayes model. For K-Nearest Neighbors, change "NB_topic_classification" to be "KNN_topic_classification")

```
categ_data.rename(columns={'Consumer complaint narrative':'Consumer_compla
int_narrative'},inplace =True)

Debt_collection = Y[:,0]
Mortgage = Y[:,1]
Credit_reporting = Y[:,2]
Credit_card = Y[:,3]
Bank_account_or_service = Y[:,4]
Consumer_Loan = Y[:,5]
Student_loan = Y[:,6]
Prepaid_card = Y[:,7]
Payday_loan = Y[:,8]
Money_transfers = Y[:,9]
Other_financial_service = Y[:,10]

predicted_Debt_collection =  NB_topic_classification(categ_data.Consumer_c
omplaint_narrative, Debt_collection)
predicted_Mortgage= NB_topic_classification(categ_data.Consumer_complaint_
narrative, Mortgage)
predicted_Credit_reporting =  NB_topic_classification(categ_data.Consumer_
complaint_narrative, Credit_reporting)
predicted_Credit_card =  NB_topic_classification(categ_data.Consumer_compl
aint_narrative, Credit_card)
predicted_Bank_account_or_service =  NB_topic_classification(categ_data.Co
nsumer_complaint_narrative, Bank_account_or_service)
predicted_Consumer_Loan =  NB_topic_classification(categ_data.Consumer_com
plaint_narrative, Consumer_Loan)
predicted_Student_loan= NB_topic_classification(categ_data.Consumer_compla
int_narrative, Student_loan)
predicted_Prepaid_card =  NB_topic_classification(categ_data.Consumer_comp
laint_narrative, Prepaid_card)
predicted_Payday_loan =  NB_topic_classification(categ_data.Consumer_compl
aint_narrative, Payday_loan)
predicted_Money_transfers =  NB_topic_classification(categ_data.Consumer_c
omplaint_narrative, Money_transfers)
predicted_Other_financial_service =  NB_topic_classification(categ_data.Co
nsumer_complaint_narrative, Other_financial_service)

X_train, X_test, y_train, y_test = train_test_split(categ_data.Consumer_co
mplaint_narrative, Y ,test_size=0.20, random_state=0)

zip_all = list(zip(predicted_Debt_collection, predicted_Mortgage, predicte
d_Credit_reporting, predicted_Credit_card, predicted_Bank_account_or_servi
ce,predicted_Consumer_Loan,predicted_Student_loan,predicted_Prepaid_card,p
redicted_Payday_loan,predicted_Money_transfers,predicted_Other_financial_s
ervice))
```

One-vs-rest models: Show accuracy metrics

```
# Explore the size of each class our dataset
agg = df.groupby('Product',as_index = False).count()
agg_sorted = agg.rename(columns={'Consumer_complaint_narrative':'Total_Num
bers'}).sort_values(by = ['Total_Numbers'])
agg_sorted

# After seeing the ranking of the size for each classes, set the rule to p
rioritize relatively smaller classes (by ranking them in ascending order)
since we can choose only one label for a document; this is done in order t
o follow One-vs-rest strategy for multi-class single-label classification

def prioritize(multilabel_list):
    if multilabel_list[7] == 1:
        return np.array([0,0,0,0,0,0,0,1,0,0,0])
    elif multilabel_list[5] == 1:
        return np.array([0,0,0,0,0,1,0,0,0,0,0])
    elif multilabel_list[8] == 1:
        return np.array([0,0,0,0,0,0,0,0,1,0,0])
    elif multilabel_list[9] == 1:
        return np.array([0,0,0,0,0,0,0,0,0,1,0])
    elif multilabel_list[10] == 1:
        return np.array([0,0,0,0,0,0,0,0,0,0,1])
    elif multilabel_list[1] == 1:
        return np.array([0,1,0,0,0,1,0,0,0,0,0])
    elif multilabel_list[0] == 1:
        return np.array([1,0,0,0,0,0,0,0,0,0,0])
    elif multilabel_list[2] == 1:
        return np.array([0,0,1,0,0,0,0,0,0,0,0])
    elif multilabel_list[3] == 1:
        return np.array([0,0,0,1,0,0,0,0,0,0,0])
    elif multilabel_list[6] == 1:
        return np.array([0,0,0,0,0,0,1,0,0,0,0])
    elif multilabel_list[4] == 1:
        return np.array([0,0,0,0,1,0,0,0,0,0,0])
    else :
        return np.array([0,0,0,0,0,0,0,0,0,0,0])

# Apply the function above to convert the response to be one-versus-rest
raw_predicted = np.array(list(zip_all))
OVR_predicted = np.array([prioritize(i) for i in raw_predicted])
OVR_predicted

# Show the accuracy metrics, i.e. overall accuracy, f-1 macro, f-1 weighted,
f-1 score for each particular class
print(classification_report\
      (y_test, OVR_predicted, target_names=mlb.classes_,digits=4))
```

References

Aggarwal, C.C. and Zhai, C., 2012. A survey of text classification algorithms. In *Mining text data* (pp. 163-222). Springer, Boston, MA.

Andrew, NG., 2018. Sequence Models: Long Short Term Memory (LSTM). MOOC offered by Coursera. Retrieved from https://www.coursera.org/lecture/nlp-sequence-models/long-short-term-memory-lstm-KXoay

Andrew, NG., 2018. Sequence Models: Recurrent Neural Network. MOOC offered by Coursera. Retrieved from https://www.coursera.org/lecture/nlp-sequence-models/recurrent-neural-network-model-ftkzt

Antoine J.-P. Tixier, 2017. Introduction to CNNs amd LSTMs for NLP.. Retrieved from https://pdfs.semanticscholar.org/ec63/21ea9ffd1ddfc9684f7f24b50ad41328ea28.pdf

Batista, D., 2018. Convolutional Neural Networks for Text Classification.

Retrieved from http://www.davidsbatista.net/blog/2018/03/31/SentenceClassificationConvNets/

Chevalier and Dina Dina Mayzlin. The effect of word of mouth on sales: Online book reviews. Journal of Marketing Research, 43(3):345–354, 2006.

Crawford, M., Khoshgoftaar, T.M., Prusa, J.D., Richter, A.N. and Al Najada, H., 2015. Survey of review spam detection using machine learning techniques. *Journal of Big Data*, *2*(1), p.23.

Da'u, A., & Salim, N., 2019. Aspect extraction on user textual reviews using multi-channel convolutional neural network. Retrieved from https://peerj.com/articles/cs-191/

Feng Zhu and Xiaoquan (Michael) Zhang. Impact of online consumer reviews on sales: The moderating role of product and consumer characteristics. Journal of Marketing, 74(2):133–148, 2010

Frank, K., 2016. Activity, Context, and Plan Recognition with Computational Causal Behaviour Models - Scientific Figure on ResearchGate. Retrieved from https://www.researchgate.net/figure/Confusion-matrix-for-multi-class-classification-The-confusion-matrix-of-a_fig7_314116591

Ghag, K. V., & Shah, K., 2018. Conceptual Sentiment Analysis Model. International Journal of Electrical and Computer Engineering (IJECE), 8(4), 2358.

Goldberg, Y. (2017). Neural network methods for natural language processing. San Rafael, CA: Morgan & Claypool.

Ghose, A. and Ipeirotis, P.G., 2007, August. Designing novel review ranking systems: predicting the usefulness and impact of reviews. In *Proceedings of the ninth international conference on Electronic commerce* (pp. 303-310). ACM.

Goodfellow, I., Bengio, Y., & Courville, A. (2017). Deep learning. Cambridge, MA: MIT Press.

Hastie, T., Tibshirani, R. J., & Friedman, J. H. (2017). The elements of statistical learning: Data mining, inference, and prediction. New York: Springer.

Hotho, A., Nürnberger, A. and Paaß, G., 2005, May. A brief survey of text mining. In *Ldv Forum* (Vol. 20, No. 1, pp. 19-62).
Hsu, C.W., Chang, C.C. and Lin, C.J., 2003. A practical guide to support vector classification.

Hutto, C.J. & Gilbert, Eric. ,2015. VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. Proceedings of the 8th International Conference on Weblogs and Social Media, ICWSM 2014.

“Introducing Deep Learning with MATLAB” (n.d.) – MathWorks, 2018. Retrieved from https://www.mathworks.com/content/dam/mathworks/tag-team/Objects/d/80879v00_Deep_Learning_ebook.pdf

James, G., Witten, D., Hastie, T. and Tibshirani, R., 2013. *An introduction to statistical learning* (Vol. 112, p. 18). New York: springer.

Joachims, T., 1998, April. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning* (pp. 137-142). Springer, Berlin, Heidelberg.

Jiayu Wu & Tianshu Ji (2016). Deep Learning for Amazon Food Review Sentiment Analysis. (n.d.). Retrieved from http://cs224d.stanford.edu/reports/WuJi.pdf

Kelleher, J. D., MacNamee, B., & DArcy, A. (2015). Fundamentals of manine learning for predictive data analytics: Algorithms, worked examples, and case studies. Cambridge, MA: The MIT Press.

Kim, S.M., Pantel, P., Chklovski, T. and Pennacchiotti, M., 2006, July. Automatically assessing review helpfulness. In *Proceedings of the 2006 Conference on* empirical methods in natural language processing (pp. 423-430). Association for Computational Linguistics.

Kim, Y. ,2014. Convolutional Neural Networks for Sentence Classification. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). doi:10.3115/v1/d14-1181

Landauer, T.K., Foltz, P.W. and Laham, D., 1998. An introduction to latent semantic analysis. *Discourse processes*, *25*(2-3), pp.259-284.
Lane, H., Howard, C., & Hapke, H. M. (2019). Natural language processing in action: Understanding, analyzing, and generating text with Python. USA: Manning Publications.

Le, Q. and Mikolov, T., 2014, January. Distributed representations of sentences and documents. In *International conference on machine learning* (pp. 1188-1196).

Lilleberg, J., Zhu, Y. and Zhang, Y., 2015, July. Support vector machines and word2vec for text classification with semantic features. In *2015 IEEE 14th International Conference on Cognitive Informatics & Cognitive Computing (ICCI* CC)* (pp. 136-140). IEEE.

Liu, Y., Huang, X., An, A. and Yu, X., 2008, December. Modeling and predicting the helpfulness of online reviews. In *2008 Eighth IEEE international conference on data mining* (pp. 443-452). IEEE.

Manning, C.D., Raghavan P., & Schütze, H., 2008. Introduction to information retrieval. Cambridge University Press.

McAuley, J.J. and Leskovec, J., 2013, May. From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews. In *Proceedings of the 22nd international conference on World Wide Web* (pp. 897-908). ACM.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S. and Dean, J., 2013a. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (pp. 3111-3119).
Mikolov, T., Chen, K., Corrado, G. and Dean, J., 2013b. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Nguyen, M., 2019. Illustrated Guide to LSTM's and GRU's: A step by step explanation. Retrieved from https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

Olson, S., R., Moore, & H., J., 2016. Identifying and Harnessing the Building Blocks of Machine Learning Pipelines for Sensible Initialization of a Data Science Automation Tool. Retrieved from https://arxiv.org/abs/1607.08878

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. and Vanderplas, J., 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research*, *12*(Oct), pp.2825-2830.

Raschka, S., & Mirjalili, V. (2017). Python machine learning: Machine learning and deep learning with Python, scikit-learn, and TensorFlow. Birmingham: Packt Publishing.

Redbord, M. (2019). The Hard Truth About Acquisition Costs (and How Your Customers Can Save You). Retrieved from https://blog.hubspot.com/news-trends/customer-acquisition-study?_ga=2.30579126.869780861.1541182844-402761219.1539702262

Rong, X., 2014. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*.

Saha, S. 2018. A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way. Retrieved from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Salehan, M. and Kim, D.J., 2016. Predicting the performance of online consumer reviews: A sentiment mining approach to big data analytics. Decision Support Systems, 81, pp.30-40.

Shojaee, S., Murad, M.A.A., Azman, A.B., Sharef, N.M. and Nadali, S., 2013, December. Detecting deceptive reviews using lexical and syntactic features. In *2013 13th International Conference on Intellient Systems Design and Applications* (pp. 53-58). IEEE.

Sokolova, M. and Lapalme, G., 2009. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, *45*(4), pp.427-437.
Thomo, A., 2009. Latent semantic analysis (Tutorial). *Victoria, Canda*, pp.1-7.

Song, G., Ye, Y., Du, X., Huang, X., & Bie, S. (2014). Short Text Classification: A Survey. Journal of Multimedia,9(5).

Trstenjak, B., Mikac, S., & Donko, D. (2014). KNN with TF-IDF based Framework for Text Categorization. Procedia Engineering,69, 1356-1364. doi:10.1016/j.proeng.2014.03.129

Vinodhini, G. and Chandrasekaran, R.M., 2012. Sentiment analysis and opinion mining: a survey. International Journal, 2(6), pp.282-292.

Yao, L., Mao, C., & Yuan, L. (2018). Graph Convolutional Networks for Text Classification. Retrieved from https://arxiv.org/abs/1809.05679

Zhang, Ye, Wallace, & Byron, 2016. A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification. Retrieved from https://arxiv.org/abs/1510.03820