# Machine Learning Engineer Nanodegree
# Capstone Project Report

Felix H.L Lou

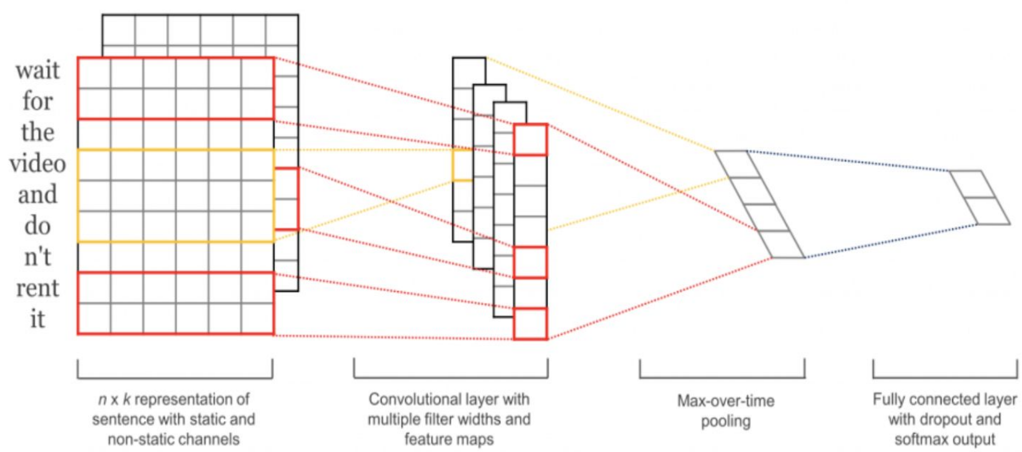
June 28th, 2018

## ***Multi-channel Convolutional Neural Network for Text Classification***

## 1. Definition

### Problem Overview

Natural Language Processing, a.k.a NLP, is a study of automatic manipulation of natural language. In other words, it focuses on the interactions between human languages and computers, specifically on how to program computers to analyze and process a huge amount of natural language data.
Deep Learning is one of the members of the Machine Learning family. Originally inspired by how the human brain functions, Deep Learning models can be enhanced by training them with more examples whilst increasing their depth (Layers), so-called Deep Neural Network.
In this project, we leveraged Convolutional Neural Networks with different kernel (gram) sizes to read customers' reviews and perform classification.




*Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification*

### Problem Statement

**a. Description**

In this project, we leveraged Deep Neural Network to perform supervised learning, in which the model learned to 'read' customers' reviews to tell the relevant sentiment. In other words, the goal was to build a model to detect customers' emotional tone in their reviews.

b. **Breakdown**
   i. Task: Classify reviews' potential sentiment (Emotional tone)
   ii. Performance: Accuracy/ F-score of the predictions on a test set
   iii. Independent variable (Predictor): Customers' review
   iv. Dependent variable (Outcome): Customers' sentiment (Positive/ Negative/ Neutral)

### Metrics

Accuracy is probably the most common metric for classification problems. For multi-class classification problems, it is not uncommon to see people using f-score, which considers both precision and recall to evaluate models' performance:

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}$$

Whether to lean towards precision or recall depends on businesses. In this project, we used f-1 score, which captures the harmonic mean of precision and recall to objectively evaluate the model's performance:

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

## 2. Analysis

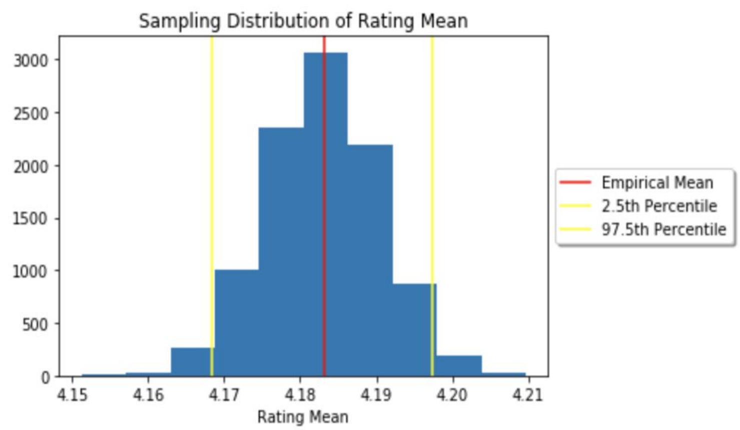
### Data Exploration

The dataset originally included 23486 observations and 10 independent variables. Each observation corresponds to a customer's review, and includes other variables. Additional columns like 'Word Count' and 'Customer Satisfaction' were created for exploratory purposes and statistical analysis. Columns regarding sentiments were also created with the help of the NLTK package. There are about 844 observations missing certain values, which were dropped when preprocessing; so the actual observations that later fed into the model was only 22628. The index column in the original csv file was also dropped.

```
Data columns (total 17 columns):
Clothing ID                 22628 non-null int64
Age                         22628 non-null int64
Title                       19662 non-null object
Review Text                 22628 non-null object
Rating                      22628 non-null int64
Recommended IND             22628 non-null int64
Positive Feedback Count     22628 non-null int64
Division Name               22628 non-null object
Department Name             22628 non-null object
Class Name                  22628 non-null object
Word Count                  22628 non-null int64
Customer Satisfaction       22628 non-null int64
Polarity Score              22628 non-null float64
Positive Score              22628 non-null float64
Neutral Score               22628 non-null float64
Negative Score              22628 non-null float64
Sentiment                   22628 non-null object
```

'Word Count' is based on 'Review Text'. And in other to get 'Customer Satisfaction', we leveraged the mean of 'Rating' to get a threshold of around 4. This was proved to be probabilistic through bootstrapping (10000 simulations):

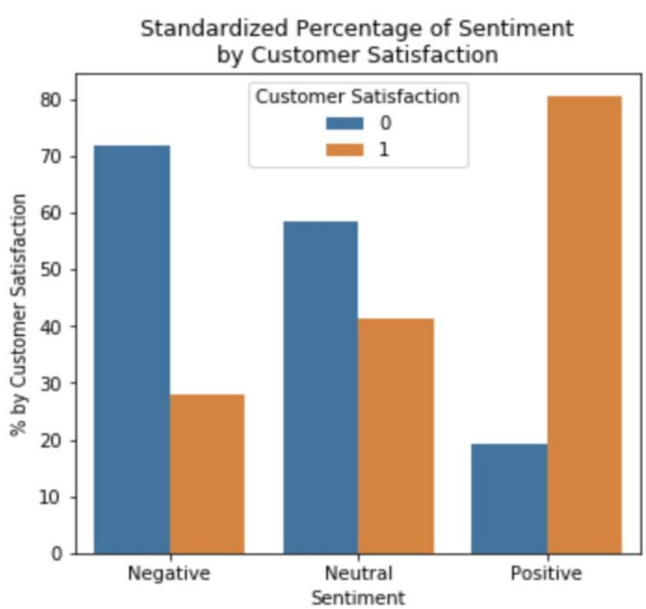
```
Empirical rating mean = 4.183091744741029
95% confidence interval = [4.16833017 4.19741029]
```




Therefore, customers with 'Rating' >= 4 were labeled as satisfied customers (1), < 4 were labeled as unsatisfied customers (0).

## Exploratory Visualization

This plot below shows that being a satisfied customer is generally a good indicator of positive sentiment, and the same logic applies for the relationship between unsatisfied customers and negative sentiment:



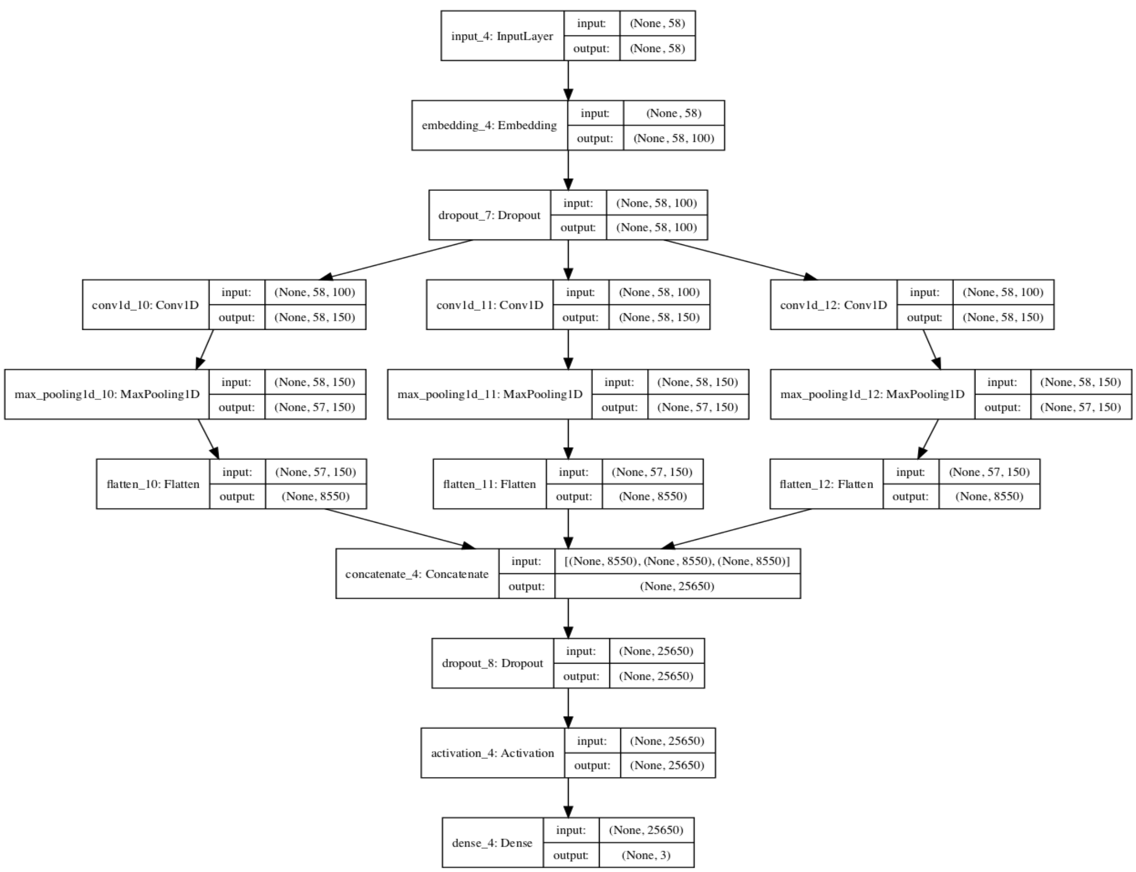

In order to get an idea of what phrases are popular among customers, we created a lookup table to visualize n-grams phrases that are likely to occur among customers. This is helpful for deciding the number of grams when creating CNN channels. Here is the lookup table of the top 10 phrases from satisfied customers:

```
Satisfied Customers Top Phrases
```

| | 2-Gram | Count | 4-Gram | Count | 6-Gram | Count | 8-Gram | Count | 10-Gram | Count |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | true size | 1203 | 26 waist 36 hips | 29 | 115 lbs 30 dd 26 waist | 10 | 34b 26 waist 36 hips hem falls inches | 3 | lightweight soft cotton shorts think meant bea... | 2 |
| **1** | love dress | 664 | looks great skinny jeans | 25 | 26 waist 36 hips hem falls | 7 | 135 lbs 35 29 36 34d long torso | 3 | realize ordered like matches look photos short... | 2 |
| **2** | looks great | 574 | 34b 26 waist 36 | 25 | tall 145 lbs 38 32 40 | 7 | wearing medium photos reference measurements 3... | 3 | outfit hot hot days loose elastic realize orde... | 2 |
| **3** | usually wear | 555 | love love love dress | 23 | 34b 26 waist 36 hips hem | 6 | tall 145 lbs 38 36d 32 40 size | 3 | hot hot days loose elastic realize ordered lik... | 2 |
| **4** | fit perfectly | 540 | 115 lbs 30 dd | 21 | 5ft2in 34b 26 waist 36 hips | 5 | adorable comfortable 115 lbs ordered xs hits k... | 2 | hae prefered colors photo big deal guess order... | 2 |
| **5** | fits perfectly | 480 | dress fits true size | 17 | regular size small 34d 27 35 | 5 | outfit hot hot days loose elastic realize ordered | 2 | hot days loose elastic realize ordered like ma... | 2 |
| **6** | size small | 427 | size small fits perfectly | 16 | 36dd 10 12 tops 12 14 | 4 | weight summer outfit hot hot days loose elastic | 2 | days loose elastic realize ordered like matche... | 2 |
| **7** | love love | 415 | lbs 30 dd 26 | 15 | photos reference measurements 38 30 40 | 4 | light weight summer outfit hot hot days loose | 2 | loose elastic realize ordered like matches loo... | 2 |
| **8** | love top | 405 | usually wear small medium | 14 | 145 lbs 38 36d 32 40 | 4 | thin light weight summer outfit hot hot days | 2 | elastic realize ordered like matches look phot... | 2 |
| **9** | usual size | 393 | great skinny jeans leggings | 14 | tall 145 lbs 38 36d 32 | 4 | wearing thin light weight summer outfit hot hot | 2 | ordered like matches look photos shorts low cu... | 2 |

## Algorithms and Techniques

The classifier we built was a Multi-Channel Convolutional Network. In this project, we created 3 channels of CNN (different grams), which helped us extract features from an (dropout) embedding layer that was created based on an embedding matrix from a pre-trained Word2Vec model. After a Max-Pooling layer, the network was concatenated, passed through a dropout layer, and ended with a dense layer to perform final interpretation:



The following hyperparameters could be tuned to potentially enhance the performance of the model:

- Optimizer Hyperparameters
    - Learning rate
    - Batch size
    - The number of training iterations (Epochs)
    - Optimization Algorithm (Update rule)
    - Dropout rate
    - Weight regularization
- Model Hyperparameters
    - Kernel size (n-grams)
    - The number of filters (Feature maps)
    - The number of embedding dimension (Word representations)
    - The number of channels (For carrying different kernels (n-grams))
    - Network depth
    - Layer types/ sequence
    - The use of pre-trained model (Word embeddings)

### Benchmark

The model's performance was compared with the top kernel of the dataset on Kaggle. The benchmark model adopts the Multinomial Naive Bayes Algorithm to make predictions. Although the person uses 'Recommended IND' and 'Rating' as target variables, the fact that it is a NLP classification problem does not change. We tested our Neural Network on the same target variables to compare the performance. For 'Recommended IND', the person's model yielded a f-1 score of 0.88. Our goal was to beat its prediction on 'Recommended IND'. Meanwhile, we wanted to see how our model did on detecting customers' sentiments (Emotional tone), which in this setting was a multi-class classification problem (Positive/ Negative/ Neutral).

## 3. Methodology

### Data Preprocessing

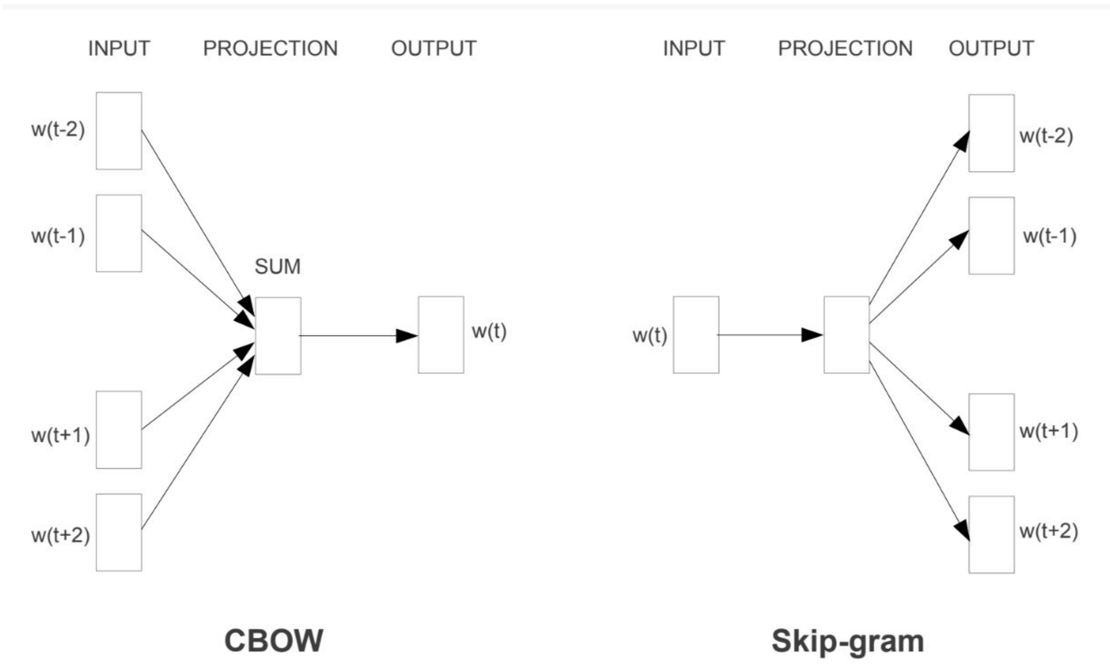
The preprocessing done in the 'Text Preprocessing, Model Building & Training' section focused on the 'Review Text' that needed for building a Word2Vec model. Before feeding the text into the Word2Vec class from Gensim, the text was cleaned by going through these steps:

1. Remove stopwords from reviews
2. Remove words that are not alphabetic
3. Remove punctuation from reviews
4. Filter out short words (Minimum character = 2)
5. Tokenize words within each review

The 'cleaned' text was then fed into the Word2Vec class to build a model. The reason why we set out to build this model in the first place was because we wanted to leverage the pre-trained word embedding to build our CNN classifier. A word embedding is a learned representation for text where words that have the same meaning have a similar representation. When developing the word embedding with Word2Vec, here were the parameters defined:

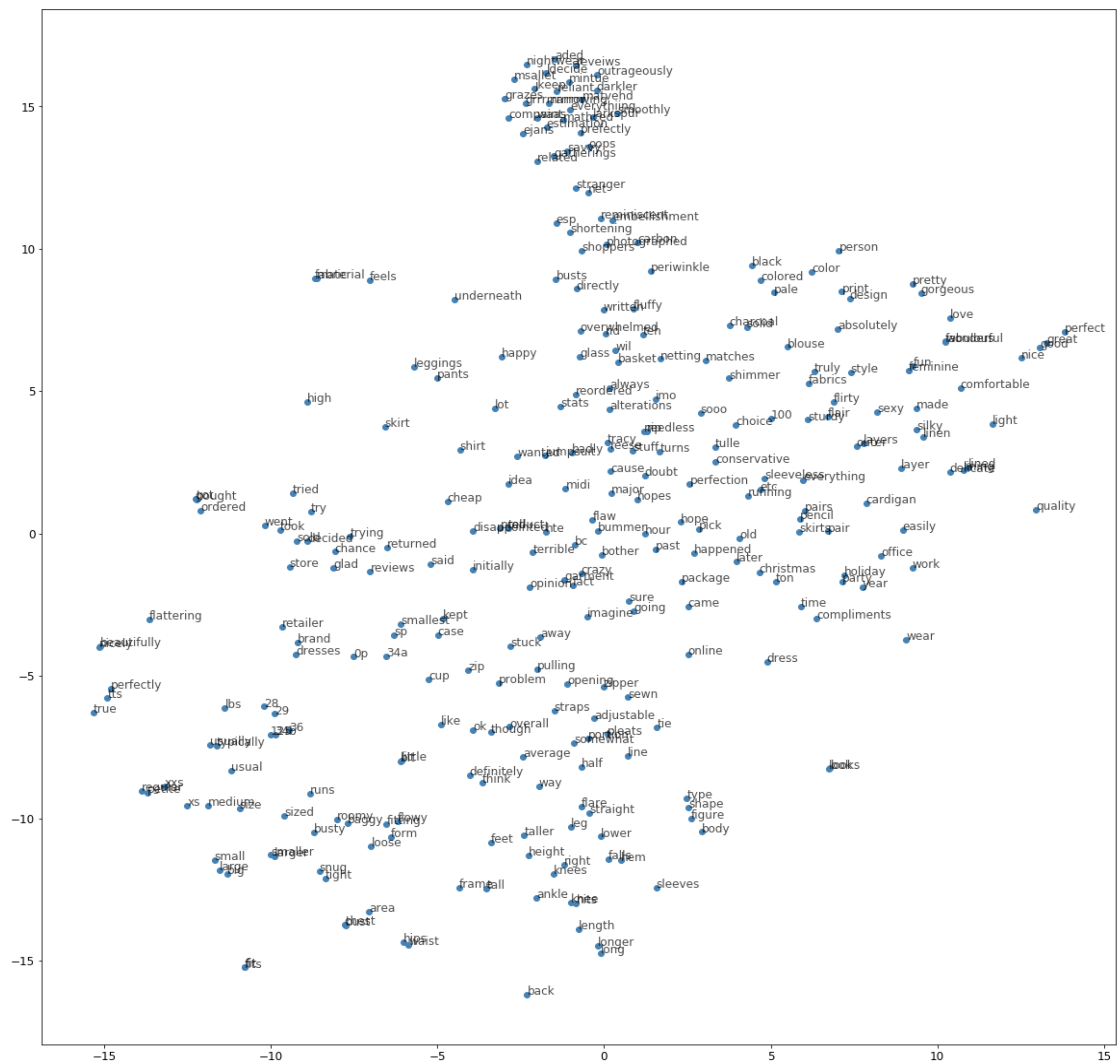
- **size**: 100 (Default) - This is the number of dimensions of the embedding. That is, the length of the dense vector to represent each token (word)
- **window**: 5 (Default) - This is the maximum distance between a target word and words around the target word
- **Min_count**: 1 - This is the minimum count of words to consider when training the model; words with an occurrence less than this count will be ignored
- **workers**: 3 (Default) - The number of threads to use while training
- **sg**: 0 (Default as CBOW) - The training algorithm, either CBOW (0) or skip-gram (1)

The CBOW (Continuous Bag-of-Words) model learns the embedding by predicting the current word based on its context. The continuous skip-gram model learns by predicting the surrounding words given a current word.




Word2Vec Training Models
Taken from "Efficient Estimation of Word Representations in Vector Space", 2013

After the model was built, it was saved so we could reuse it when building our CNN classifier afterwards. The way we leveraged it was to extract its weights to construct an embedding matrix, which would be used to build the embedding layer in our CNN classifier. Word2Vec is a statistical method for efficiently learning a standalone word embedding from a text corpus, and it could make the neural network based training of the embedding more efficient. Here is a plot of the high dimension word vectors cluster (First 300 words) learn by the model:




## Implementation

The implementation process can generally be divided into 2 stages:

1. Encode text data
2. Build CNN model

In the first stage, we first made use of the sklearn function to split the data set into train and test set. We then cleaned the train set and encode the data with the helper functions defined in the scripts of 'nlp.py' and 'cnn.py' respectively. For predicting sentiment types, we needed to numerically encode and categorize the labels beforehand:

1. Numerically encode and categorize labels (For predicting sentiments)
2. Split the data set, with 'Review_Text' as inputs and 'Recommended IND' or 'Sentiments' as labels (Outputs)
3. clean_reviews(): apply the helper function from 'nlp.py' to clean the each set

4. create_tokenizer(): apply the helper function from ‘cnn.py’ to create a tokenizer on the train set
5. max_length(): apply the helper function from ‘cnn.py’ to get the maximum review length
6. encode_reviews(): apply the helper function from ‘cnn.py’ to encode and pad all reviews in each set

In the second stage, we defined the model’s hyperparameters, built the model with the function from the script ‘cnn.py’, fit our model and logged the performance, and finally saved the trained model for testing afterwards. In particular, we specify whether or not to use a pre-trained embedding when calling the helper function to build the CNN. Here, we used the pre-trained model mentioned in the Data Preprocessing section:

1. Define model’s hyperparameters (Load a pre-trained embedding)
2. build_cnn(): apply the helper function from ‘cnn.py’ to create a multi-channel(s) review classifier
3. Fit the model and start training
4. Save the trained model, and test it on the train set

Here is a summary of our compiled model on predicting customers’ sentiments:

```
________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
================================================================================================
input_3 (InputLayer)            (None, 58)           0
________________________________________________________________________________________________
embedding_3 (Embedding)         (None, 58, 100)      1193500     input_3[0][0]
________________________________________________________________________________________________
dropout_5 (Dropout)             (None, 58, 100)      0           embedding_3[0][0]
________________________________________________________________________________________________
conv1d_7 (Conv1D)               (None, 58, 150)      30150       dropout_5[0][0]
________________________________________________________________________________________________
conv1d_8 (Conv1D)               (None, 58, 150)      60150       dropout_5[0][0]
________________________________________________________________________________________________
conv1d_9 (Conv1D)               (None, 58, 150)      75150       dropout_5[0][0]
________________________________________________________________________________________________
max_pooling1d_7 (MaxPooling1D)  (None, 57, 150)      0           conv1d_7[0][0]
________________________________________________________________________________________________
max_pooling1d_8 (MaxPooling1D)  (None, 57, 150)      0           conv1d_8[0][0]
________________________________________________________________________________________________
max_pooling1d_9 (MaxPooling1D)  (None, 57, 150)      0           conv1d_9[0][0]
________________________________________________________________________________________________
flatten_7 (Flatten)             (None, 8550)         0           max_pooling1d_7[0][0]
________________________________________________________________________________________________
flatten_8 (Flatten)             (None, 8550)         0           max_pooling1d_8[0][0]
________________________________________________________________________________________________
flatten_9 (Flatten)             (None, 8550)         0           max_pooling1d_9[0][0]
________________________________________________________________________________________________
concatenate_3 (Concatenate)     (None, 25650)        0           flatten_7[0][0]
                                                                 flatten_8[0][0]
                                                                 flatten_9[0][0]
________________________________________________________________________________________________
dropout_6 (Dropout)             (None, 25650)        0           concatenate_3[0][0]
________________________________________________________________________________________________
activation_3 (Activation)       (None, 25650)        0           dropout_6[0][0]
________________________________________________________________________________________________
dense_3 (Dense)                 (None, 3)            76953       activation_3[0][0]
================================================================================================
Total params: 1,435,903
Trainable params: 1,435,903
Non-trainable params: 0
```

### Refinement

Apart from predicting customers' sentiments, as mentioned previously, we also wanted to beat the model created in the top kernel, which has an f-1 score of 0.88 on predicting 'Recommend IND'. Our initial model yielded an accuracy of 0.85 over 10 training epochs. After a couple of trials and refinements, the general findings on enhancing the model's performance are:

- Kernel (Different filter size/ n-grams) sizes matter a lot
- Feature map sizes matter a lot
- Dropout layer seems to have little impact
- Where to insert Max-Pooling layer matters
- Model complexity (The number of layers)
- Batch size matters

Our final model yielded an accuracy of 0.91 on predicting 'Recommended IND'. Here is the training log on 'Recommended IND' over 10 epochs:

```
Epoch 1/10
15839/15839 [==============================] - 33s 2ms/step - loss: 2.1297 - acc: 0.8246
Epoch 2/10
15839/15839 [==============================] - 28s 2ms/step - loss: 0.4021 - acc: 0.8516
Epoch 3/10
15839/15839 [==============================] - 28s 2ms/step - loss: 0.3728 - acc: 0.8631
Epoch 4/10
15839/15839 [==============================] - 33s 2ms/step - loss: 0.3344 - acc: 0.8787
Epoch 5/10
15839/15839 [==============================] - 30s 2ms/step - loss: 0.3245 - acc: 0.8828
Epoch 6/10
15839/15839 [==============================] - 30s 2ms/step - loss: 0.3051 - acc: 0.8918
Epoch 7/10
15839/15839 [==============================] - 42s 3ms/step - loss: 0.2780 - acc: 0.9010
Epoch 8/10
15839/15839 [==============================] - 28s 2ms/step - loss: 0.2757 - acc: 0.9025
Epoch 9/10
15839/15839 [==============================] - 26s 2ms/step - loss: 0.2590 - acc: 0.9102
Epoch 10/10
15839/15839 [==============================] - 32s 2ms/step - loss: 0.2587 - acc: 0.9133
```

## 4. Results

### Model Evaluation and Validation

The final model architecture and hyperparameters, as illustrated in the 'Algorithms and Techniques' section, were chosen based on its performance compared with the benchmark model. The predictions on 'Sentiments' were actually better than what we expected. For both types of predictions, we generally used the same set of hyperparameters, except for activation function, loss function, and output units, as the predictions on 'Sentiments' is a multi-class problem. Here is the list of our definitions on hyperparameters:

- Embedding dimension: 100
- The number of CNN channels: 3
- Filter size (ngrams): 2, 4, 5
- Feature map size: 150
- Drop out rate: 0.5
- Weight regularization: L2 (0.03)
- Optimizer: Adam

- Batch size: 75
- The number of training epochs: 10
- Pre-trained embedding: Word2Vec on all reviews

To verify the model's robustness, it was tested on the test set, which consisted of 6789 observations that the model had never seen before. Here are the classification summary on 'Recommended IND' and 'Sentiments' respectively:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Not Recommended | 0.75 | 0.62 | 0.67 | 1167 |
| Recommended | 0.92 | 0.96 | 0.94 | 5622 |
| avg / total | 0.89 | 0.90 | 0.89 | 6789 |

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Positive | 0.94 | 0.99 | 0.97 | 6315 |
| Negative | 0.63 | 0.17 | 0.27 | 436 |
| Neutral | 0.00 | 0.00 | 0.00 | 38 |
| avg / total | 0.92 | 0.93 | 0.92 | 6789 |

### Justification

As shown in the previous section, our model yielded a f-1 score of 0.89 and 0.92 on classifying 'Recommended IND' and 'Sentiments' respectively. For 'Recommended IND', we successfully beat the benchmark model by a slightly amount of 0.01. For 'Sentiments', the classification performance was even better; so we were quite successful in the scope of this project.
To caveat, the results were based on a relatively small data set. To recall, we only had 22628 observations after cleaning the data set. And the word embedding was trained on about 14000 vocabularies. In reality, embeddings are learned from much larger corpora of text, which sometimes could be billions of words. In a nutshell, if we wanted to deploy the model on the e-commerce site for classifying customers' reviews, we'd probably need more data and let our model 'study' more text.

## 5. Conclusion

### Free Form Visualization

We wanted to see how the model performed without the presence of a pre-trained embedding; so we went ahead and trained the model with the same set of hyperparameters on classifying 'Sentiments' without a pre-trained (Word2Vec) )embedding. Here are the training log and the model's final performance on the test set:

```
Epoch 1/10
15839/15839 [==============================] - 32s 2ms/step - loss: 1.5335 - acc: 0.9288
Epoch 2/10
15839/15839 [==============================] - 33s 2ms/step - loss: 0.2205 - acc: 0.9319
Epoch 3/10
15839/15839 [==============================] - 30s 2ms/step - loss: 0.1915 - acc: 0.9412
Epoch 4/10
15839/15839 [==============================] - 29s 2ms/step - loss: 0.1640 - acc: 0.9522
Epoch 5/10
15839/15839 [==============================] - 32s 2ms/step - loss: 0.1483 - acc: 0.9593
Epoch 6/10
15839/15839 [==============================] - 28s 2ms/step - loss: 0.1354 - acc: 0.9629
Epoch 7/10
15839/15839 [==============================] - 27s 2ms/step - loss: 0.1222 - acc: 0.9685
Epoch 8/10
15839/15839 [==============================] - 28s 2ms/step - loss: 0.1178 - acc: 0.9692
Epoch 9/10
15839/15839 [==============================] - 35s 2ms/step - loss: 0.1055 - acc: 0.9729
Epoch 10/10
15839/15839 [==============================] - 28s 2ms/step - loss: 0.0966 - acc: 0.9766
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Positive | 0.95 | 0.96 | 0.96 | 6315 |
| Negative | 0.41 | 0.36 | 0.38 | 436 |
| Neutral | 0.00 | 0.00 | 0.00 | 38 |
| avg / total | 0.91 | 0.92 | 0.92 | 6789 |

As shown above, there was not much of a difference regarding the model's final performance on the test set. In fact, the model was actually learning faster in the sense of achieving higher accuracy during the training phase.

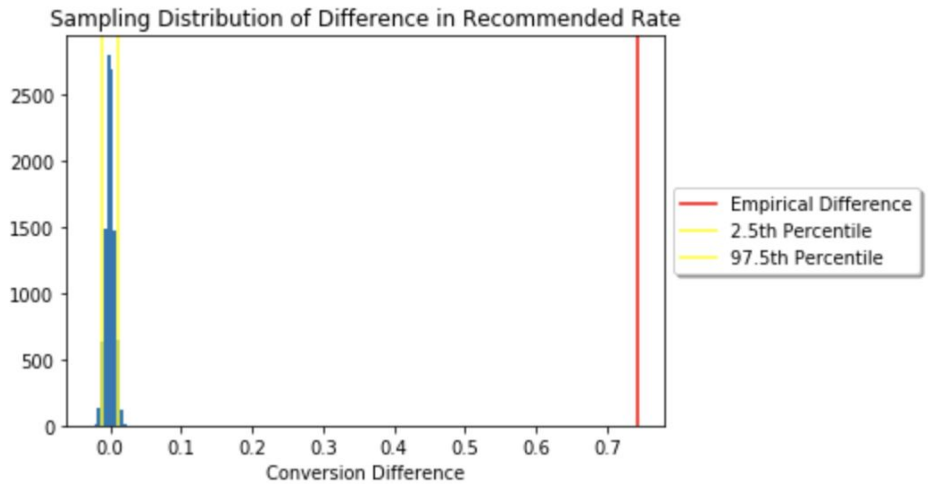
## Reflection

The entire process used in this project could be summarized by the following:

1. Discover a problem, and define the scope of the project (How to approach the problem)
2. Data gathering
3. Setting a benchmark that the model can be compared with
4. Preliminary investigation of the data set
5. Text preprocessing and model building
6. Model training and refinement
7. Model testing and refinement

As with most projects we've done before, interesting insights usually start to show up early in the process. Like in the preliminary investigation phase in this project, we got to discover a 'Rating' of 4 is a threshold between satisfied and unsatisfied customers, which in turn is actually a very good indicator of customers' sentiments. We went on to check the recommended rate of these customers, and found that the rates (Conversions) are 0.99 and 0.25 for satisfied and unsatisfied customers respectively. The difference was even proved significant based on permutation (10000 simulations), which helped us reject the null hypothesis that there is no difference in recommended rate between satisfied and unsatisfied customers:

Empirical Recommended Rate Difference = 0.743




Again, with most Machine Learning problems, hyperparameters tuning is as much a science as an art. That is why we found steps 5-7 the most challenging. It took patience and trial-and-error to yield good results because at the end of the day, how to define the model and values really depends on what sort of problem we are dealing with. Having that said, this is what makes Machine Learning interesting!

### Improvement

As mentioned previously, one possible way to enhance the model's performance is to train it on a larger and more representative data set. This would mean a data set that includes more observations and a much larger corpora of text (For embeddings). Regarding model architecture and complexity, here are a couple of aspects that we could think of:

- N-grams: This is the kernel size/ filter size in the CNN. Different problems supposedly require different combinations of n-grams
- The number of CNN channels: The number of channels could have great impact on performance
- Network depth: Although not necessary, deeper model usually perform better

If we were to extend the project, we'd like to try deeper network or perhaps construct CNNs on character level. This would mean greater computational power and labor-intensive effort (For text preprocessing) are required.

## 6. Reference


- Convolutional Neural Networks for Sentence Classification
- Convolutional Neural Networks for Sentence Classification (code)
- Best Practices for Document Classification with Deep Learning
- Character-level Convolutional Networks for Text Classification
- A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification
- Natural Language Processing (almost) from Scratch

- A Primer on Neural Network Models for Natural Language Processing
- A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts
- Understanding Convolutional Neural Networks for NLP
- Guided Numeric and Text Exploration E-Commerce
- List of NLP resources
- Word Cloud with Python
- Deep Learning for NLP Best Practices
- Embed, encode, attend, predict: The new deep learning formula for state-of-the-art NLP models