Import Libraries

In [1]:
```
# Python Version: 3.7.9
# Pandas Version: 1.3.5
# Beautiful4 Soup Version: 4.11.1
# Contractions Version: 0.0.18
# Setuptools Version: 60.2.0
# Symspellpy Version: 6.7.7
# NLTK Version: 3.8.1
# Scikit-learn Version: 1.0.2

import pandas as pd
import re
from bs4 import BeautifulSoup
import contractions as ct
import pkg_resources
from symspellpy import SymSpell
import warnings

from nltk.corpus import wordnet as wn
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk import map_tag, WordNetLemmatizer, pos_tag

from sklearn.svm import LinearSVC
from sklearn.metrics import classification_report
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import Perceptron, LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

Define Functions

In [2]:
```
# Drop empty & duplicated rows
def init_data(data_frame):
    data_frame.dropna(inplace=True)
    data_frame.drop_duplicates(inplace=True)
    data_frame['star_rating'] = data_frame['star_rating'].astype('int')
    return data_frame

# Init spell checker object
def init_spell_checker():
    sym_spell_obj = SymSpell(max_dictionary_edit_distance=2, prefix_length=7)
    dictionary_path = pkg_resources.resource_filename(
        "symspellpy", "frequency_dictionary_en_82_765.txt"
    )
    bigram_path = pkg_resources.resource_filename(
        "symspellpy", "frequency_bigramdictionary_en_243_342.txt"
    )
    sym_spell_obj.load_dictionary(dictionary_path, term_index=0, count_index=1)
    sym_spell_obj.load_bigram_dictionary(bigram_path, term_index=0, count_index=2)

    return sym_spell_obj

# Spell correct the input text
def spell_correct(text):
    input_term = text
    suggestions = sym_spell.lookup_compound(
        input_term, max_edit_distance=2, transfer_casing=True
    )
    return suggestions[0].term

# Lemmatize the input word
def word_lemmatization(word):
    treebank_pos_tag = pos_tag([word])[0][1]
    universal_pos_tag = map_tag('en-ptb', 'universal', treebank_pos_tag)

    if universal_pos_tag == "ADJ":
        word = wnl.lemmatize(word, wn.ADJ)
    elif universal_pos_tag == "VERB":
        word = wnl.lemmatize(word, wn.VERB)
    elif universal_pos_tag == "NOUN":
        word = wnl.lemmatize(word, wn.NOUN)
    elif universal_pos_tag == "ADV":
        word = wnl.lemmatize(word, wn.ADV)
        word = get_adverb_lemma(word)

    return word

# Get an input adverb's lemma if any
def get_adverb_lemma(word):
    has_suggestion = False
    param_wn_synset = word + ".r.01"

    # check if word's synset contains adverb option
    for i in range(0, len(wn.synsets(word))):
        if param_wn_synset == str((wn.synsets(word)[i])).split("\'")[1]:
            has_suggestion = True

    if not has_suggestion:
```

```python
        return word

    # if yes and suggestion not empty then return suggestion, else return original word
    suggest_lemma_list = wn.synset(param_wn_synset).lemmas()[0].pertainyms()
    if len(suggest_lemma_list) > 0:
        return suggest_lemma_list[0].name()
    else:
        return word

# Remove stop words and lemmatize the remaining words
def lemmatize_non_stopwords(review_body_string):
    word_tokens = word_tokenize(review_body_string)
    buffer_string = ""

    for w in word_tokens:
        if w not in stop_words:
            w = word_lemmatization(w)
            buffer_string = buffer_string + w + "  "

    buffer_string = re.sub(' +', ' ', buffer_string).strip()
    return buffer_string

# Data cleaning & preprocessing
def data_cleaning(data_frame):
    before_data_cleaning_reviews_total_length = 0
    after_data_cleaning_reviews_total_length = 0
    before_data_preprocessing_reviews_total_length = 0
    after_data_preprocessing_reviews_total_length = 0

    for i in range(0, len(data_frame)):

        if data_frame['star_rating'][i] == '1' or data_frame['star_rating'][i] == '2':
            data_frame.loc[i, ['star_rating']] = 'Class 1'
        elif data_frame['star_rating'][i] == '3':
            data_frame.loc[i, ['star_rating']] = 'Class 2'
        elif data_frame['star_rating'][i] == '4' or data_frame['star_rating'][i] == '5':
            data_frame.loc[i, ['star_rating']] = 'Class 3'

        review_text = data_frame['review_body'][i]
        before_data_cleaning_reviews_total_length = before_data_cleaning_reviews_total_length + len(review_text)

        # remove un-wanted html tags
        if BeautifulSoup(review_text, "html.parser").find():
            review_text = BeautifulSoup(review_text, "html.parser").get_text("  ")

        # spell correction
        review_text = spell_correct(review_text)

        # text extend contractions
        review_text = ct.fix(review_text)

        # remove non-alphabetical chars
        regex = re.compile('[^a-zA-Z]')
        review_text = regex.sub(' ', review_text)

        # convert to lower case
        review_text = review_text.lower().strip()
        review_text = " ".join(review_text.split())

        # end of data cleaning, before data processing
        after_data_cleaning_reviews_total_length = after_data_cleaning_reviews_total_length + len(review_text)

        # start of data processing
        before_data_preprocessing_reviews_total_length = before_data_preprocessing_reviews_total_length + len(
            review_text)
        review_text = lemmatize_non_stopwords(review_text)
        # end of data processing
        review_text = " ".join(review_text.split())
        after_data_preprocessing_reviews_total_length = after_data_preprocessing_reviews_total_length + len(review_text)

        data_frame.loc[i, ['review_body']] = review_text

    print("Average length of reviews before data cleaning: " + str(before_data_cleaning_reviews_total_length / len(
        data_frame)) + ", Average length of reviews after data cleaning: " + str(
        after_data_cleaning_reviews_total_length / len(data_frame)))
    print("Average length of reviews before data preprocessing: " + str(
        before_data_preprocessing_reviews_total_length / len(
            data_frame)) + ", Average length of reviews after data preprocessing: " + str(
        after_data_preprocessing_reviews_total_length / len(data_frame)))
    print("\n")

    return data_frame

# Print the training result
def generate_report(y_test, y_pred):
    report = classification_report(y_test, y_pred, zero_division=1, output_dict=True)
    print("Class 1 Precision: " + str(report['Class 1']['precision']) + ", Class 1 Recall: " + str(
        report['Class 1']['recall']) + ", Class 1 f1-score: " + str(report['Class 1']['f1-score']))
    print("Class 2 Precision: " + str(report['Class 2']['precision']) + ", Class 2 Recall: " + str(
        report['Class 2']['recall']) + ", Class 2 f1-score: " + str(report['Class 2']['f1-score']))
    print("Class 3 Precision: " + str(report['Class 3']['precision']) + ", Class 3 Recall: " + str(
        report['Class 3']['recall']) + ", Class 3 f1-score: " + str(report['Class 3']['f1-score']))
    print("Average Precision: " + str(report['macro avg']['precision']) + ", Averagage Recall: " + str(
        report['macro avg']['recall']) + ", Averagage f1-score: " + str(
        report['macro avg']['f1-score']))
    print("\n")
```

Initialization

In [3]:
```
# Init
RANDOM_SAMPLE_SIZE = 20000
warnings.filterwarnings("ignore", category=UserWarning, module='bs4')
sym_spell = init_spell_checker()
stop_words = set(stopwords.words('english'))
wnl = WordNetLemmatizer()
```

Read Data

In [4]:
```
# Reading data from cache. (data.pkl was generated by reading the given Amazon's dataset provided in HW1 description)
df = pd.read_pickle("./data.pkl")
df = init_data(df).reset_index(drop=True)
```

Form three classes and select 20000 reviews randomly from each class.

In [5]:
```
 # 3-classes and concat into 1
class1_df = df[df['star_rating'] <= 2].sample(RANDOM_SAMPLE_SIZE)
class2_df = df[df['star_rating'] == 3].sample(RANDOM_SAMPLE_SIZE)
class3_df = df[df['star_rating'] >= 4].sample(RANDOM_SAMPLE_SIZE)

balanced_df = pd.concat([class1_df, class2_df, class3_df]).reset_index(drop=True)
balanced_df['star_rating'] = balanced_df['star_rating'].astype('string')
```

Data Cleaning & Pre-processing

1. Remove un-wanted Html tags

2. Spell corrections

3. Text contractions

4. Remove non-alphabetical chars

5. Convert to lower cases

6. Remove stop words

7. Lemmatisation

In [6]:
```
cleaned_balanced_df = data_cleaning(balanced_df)
 # cleaned_balanced_df cache
cleaned_balanced_df.to_pickle('cleaned_balanced_df_official.pkl')
cleaned_balanced_df = pd.read_pickle("./cleaned_balanced_df_official.pkl")
```

```
Average length of reviews before data cleaning: 278.5658833333333, Average length of reviews after data cleaning: 269.23761666666667
Average length of reviews before data preprocessing: 269.23761666666667, Average length of reviews after data preprocessing: 154.189283333333
32
```

TF-IDF Feature Extraction

In [7]:
```
# tf-idf feacture matrix
tf_idf = TfidfVectorizer(lowercase=False, ngram_range=(1, 5))
tf_idf_result = tf_idf.fit_transform(cleaned_balanced_df['review_body'])
```

Split dataset into Training and Testing Set

In [8]:
```
 X_train, X_test, y_train, y_test = train_test_split(tf_idf_result, cleaned_balanced_df['star_rating'], test_size=0.2)
```

Perceptron

In [9]:
```
# Train Perceptron Model & generate training report
clf_perceptron = Perceptron()
clf_perceptron = clf_perceptron.fit(X_train, y_train)
y_pred_perceptron = clf_perceptron.predict(X_test)
generate_report(y_test, y_pred_perceptron)
```

```
Class 1 Precision: 0.6571160169093471, Class 1 Recall: 0.6856162705219309, Class 1 f1-score: 0.671063676699844
Class 2 Precision: 0.5603917301414582, Class 2 Recall: 0.5175879396984925, Class 2 f1-score: 0.5381400208986415
Class 3 Precision: 0.707083128381702, Class 3 Recall: 0.7298806803757298, Class 3 f1-score: 0.7183010618363522
Average Precision: 0.6415302918108358, Averagage Recall: 0.6443616301987177, Averagage f1-score: 0.6425015864782794
```

SVM

In [10]:
```
# Train SVM Linear Model & generate training report
clf_linear_svc = LinearSVC(loss='hinge')
clf_linear_svc = clf_linear_svc.fit(X_train, y_train)
y_pred_linear_svc = clf_linear_svc.predict(X_test)
generate_report(y_test, y_pred_linear_svc)
```

```
Class 1 Precision: 0.7024064808196331, Class 1 Recall: 0.7223719676549866, Class 1 f1-score: 0.7122493355883065
Class 2 Precision: 0.6075845012366035, Class 2 Recall: 0.5555276381909547, Class 2 f1-score: 0.5803911274445465
Class 3 Precision: 0.7346301633045149, Class 3 Recall: 0.7765930439197766, Class 3 f1-score: 0.755029001604344
Average Precision: 0.6815403817869171, Averagage Recall: 0.6848308832552393, Averagage f1-score: 0.682556488212399
```

Logistic Regression

In [11]:
```
# Train Logistic Regression Model & generate training report
clf_logistic_regression = LogisticRegression(solver='sag')
clf_logistic_regression = clf_logistic_regression.fit(X_train, y_train)
y_pred_logistic_regression = clf_logistic_regression.predict(X_test)
generate_report(y_test, y_pred_logistic_regression)
```

```
Class 1 Precision: 0.700441609421001, Class 1 Recall: 0.699583435432492, Class 1 f1-score: 0.7000122594090965
Class 2 Precision: 0.5834348355663824, Class 2 Recall: 0.6017587939698492, Class 2 f1-score: 0.5924551638837353
Class 3 Precision: 0.7512437810945274, Class 3 Recall: 0.7283574511297284, Class 3 f1-score: 0.7396236143335911
Average Precision: 0.6783734086939702, Averagage Recall: 0.6765665601773566, Averagage f1-score: 0.6773636792088076
```

Naive Bayes

In [12]:
```
# Train MultinomialNB Model & generate training report
clf_multinomial_nb = MultinomialNB(fit_prior=False)
clf_multinomial_nb = clf_multinomial_nb.fit(X_train, y_train)
y_pred_multinomial_nb = clf_multinomial_nb.predict(X_test)
generate_report(y_test, y_pred_multinomial_nb)
```

```
Class 1 Precision: 0.7267833109017496, Class 1 Recall: 0.6616025483950012, Class 1 f1-score: 0.6926629040533607
Class 2 Precision: 0.573621103117506, Class 2 Recall: 0.6010050251256281, Class 2 f1-score: 0.5869938650306749
Class 3 Precision: 0.7297691373025517, Class 3 Recall: 0.7623762376237624, Class 3 f1-score: 0.7457164142041223
Average Precision: 0.6767245171072691, Averagage Recall: 0.6749946037147971, Averagage f1-score: 0.675124394429386
```

Authorship

Author: Che Wei Wu Date: Jan 24, 2023 Description: The source code for USC_CSCI544_SPRING23_HW1