Python Version - 3.7.0

Packages required for executing the code

In [1]:
```
# ! pip install bs4
# ! pip install contractions
# ! pip install pandas
# ! pip install numpy
# ! pip install matplotlib
# ! pip install emoji
# ! pip install textblob
# ! pip install unicodedata2
# ! pip install nltk
# ! pip install scikit-learn

# nltk.download('wordnet')
# nltk.download('omw-1.4')
# nltk.download('words')
```

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import re
from bs4 import BeautifulSoup
import contractions
import emoji
from textblob import TextBlob
from textblob import Word
from unicodedata import normalize

import nltk
from nltk.corpus import words
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from nltk.stem import PorterStemmer
from nltk import word_tokenize, pos_tag
from nltk.tokenize import sent_tokenize

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Perceptron
from sklearn.metrics import classification_report
from sklearn.preprocessing import MaxAbsScaler
from sklearn.feature_selection import SelectKBest

pd.options.display.max_colwidth = 500
import warnings
warnings.filterwarnings("ignore")
```

Dataset Preparation

Read Data

Data is read from data.tsv file using pandas read_csv function

In [3]:
```
filepath =  "data.tsv"
amazon_reviews = pd.read_csv(filepath, sep = "\t",
                             skip_blank_lines = True,
                             error_bad_lines=False,
                             warn_bad_lines = False)
```

Keep Reviews and Ratings

Drop all columns other than review_body and ratings.

In [4]:
```
amazon_reviews.drop(columns=['marketplace', 'customer_id',
                             'review_id', 'product_id',
                             'product_title', 'product_category',
                             'helpful_votes', 'total_votes',
                             'vine', 'verified_purchase',
                             'review_headline', 'review_date',
                             'product_parent'], inplace = True)

amazon_reviews.rename(columns={'review_body': 'reviews',
                               'star_rating': 'star_rating'}
                      , inplace=True)
```

NOTE:

Data sampling to keep 20000 samples from each class is performed after data cleaning to avoid nan values in review body. Train and test split is performed after generating tf-idf features

Average length of Reviews before Data Cleaning

In [5]:
```
amazon_reviews['review_length'] = amazon_reviews['reviews'].str.len()
average_len_before_dc = amazon_reviews["review_length"].mean()
print("Average Length of Review before Data Cleaning: "
      + str(average_len_before_dc))
amazon_reviews.drop(columns=['review_length'], inplace = True)
```

```
Average Length of Review before Data Cleaning: 176.38960904441382
```

Data Cleaning

Data cleaning techniques that have been used are listed below -

1. Drop data that have nan values in star_rating or reviews column
2. Convert star_rating column to integer
3. Convert reviews to lower case
4. Remove html tags
5. Remove external links and urls
6. Remove extra spaces
7. Remove accents
8. Process Emojis
9. Expand Contractions
10. Remove non alphabetical characters
11. Spell correction

Drop data that have nan values in star_rating or reviews column

In [6]:
```
amazon_reviews = amazon_reviews[amazon_reviews['star_rating'].notna()]
amazon_reviews = amazon_reviews[amazon_reviews['reviews'].notna()]
```

Convert star_rating column to integer

star_rating column has values 1, 2, 3, 4, 5, "1", "2", "3", "4", "5". All values are being converted to integers.

In [7]:
```
amazon_reviews['star_rating'] = pd.to_numeric(
    amazon_reviews['star_rating']).astype('Int64')
```

Convert reviews to lower case

In [8]:
```
reviews_sampled = amazon_reviews.copy()
reviews_sampled['reviews'] = reviews_sampled['reviews'].str.lower()
```

Remove html tags

Regex checks for reviews containing string characters within such braces <> and replaces with white space. Example -
<\html>, <\br>

In [9]:
```
reviews_sampled['reviews'] = reviews_sampled['reviews'].str.replace(
    r'<[^<>]*>', ' ', regex=True)
```

Remove external links and urls

Regex checks for reviews containing string characters with http and replaces with white space

In [10]:
```
reviews_sampled['reviews'] = reviews_sampled['reviews'].apply(
    lambda x: re.sub(r'http\S+', ' ', str(x)))
```

Remove extra spaces

All extra spaces, white spaces, tabs etc are reduced down to single white space

In [11]:
```
reviews_sampled['reviews'] = reviews_sampled['reviews'].str.strip()
reviews_sampled['reviews'] = reviews_sampled['reviews'].replace(
    r'\s+', ' ', regex=True)
```

Remove accents

Removes the accents from a string, converting them to their corresponding non-accented ASCII characters. Example -

à, á, â, ã, ä, å -> a

Any of the characters on left will be changed to "a" after accents removal

In [12]:
```
remove_accent = lambda text: normalize("NFKD", text).encode(
    "ascii", "ignore").decode("utf-8", "ignore")
reviews_sampled["reviews"] = reviews_sampled["reviews"].apply(
    remove_accent)
```

Process emojis

Emoji python package is used to convert emojis to text
Example -
👍 - thumbs up

Some text emojis are converted using regex matching and replace
Example -
:) - Happy
:(- sad, angry

In [13]:
```
reviews_sampled['reviews'] = reviews_sampled['reviews'].apply(
    lambda x: emoji.demojize(x))

reviews_sampled["reviews"].replace(to_replace=r';\(',
                                   value='sad, angry',
                                   regex=True, inplace = True)
reviews_sampled["reviews"].replace(to_replace=r';-\(',
                                   value='sad, angry',
                                   regex=True, inplace = True)
reviews_sampled["reviews"].replace(to_replace=r':\(',
                                   value='sad, angry',
                                   regex=True, inplace = True)
reviews_sampled["reviews"].replace(to_replace=r':-\(',
                                   value='sad, angry',
                                   regex=True, inplace = True)
reviews_sampled["reviews"].replace(to_replace=r':\'\(',
                                   value='sad, angry',
                                   regex=True, inplace = True)
reviews_sampled["reviews"].replace(to_replace=r';\'\(',
                                   value='sad, angry',
                                   regex=True, inplace = True)
reviews_sampled["reviews"].replace(to_replace=r';*\(',
                                   value='sad, angry',
                                   regex=True, inplace = True)


reviews_sampled["reviews"].replace(to_replace=r':\)',
                                   value='Happy', regex=True,
                                   inplace = True)
reviews_sampled["reviews"].replace(to_replace=r':\'\)',
                                   value='Happy', regex=True,
                                   inplace = True)
reviews_sampled["reviews"].replace(to_replace=r';\)',
                                   value='Happy', regex=True,
                                   inplace = True)
reviews_sampled["reviews"].replace(to_replace=r':-\)',
                                   value='Happy', regex=True,
                                   inplace = True)
reviews_sampled["reviews"].replace(to_replace=r';-\)',
                                   value='Happy', regex=True,
                                   inplace = True)
reviews_sampled["reviews"].replace(to_replace=r';-\)',
                                   value='Happy', regex=True,
                                   inplace = True)
reviews_sampled["reviews"].replace(to_replace=r':o\)',
                                   value='Happy', regex=True,
                                   inplace = True)
reviews_sampled["reviews"].replace(to_replace=r';o\)',
                                   value='Happy', regex=True,
                                   inplace = True)
reviews_sampled["reviews"].replace(to_replace=r';0\)',
                                   value='Happy', regex=True,
                                   inplace = True)
reviews_sampled["reviews"].replace(to_replace=r';O\)',
                                   value='Happy', regex=True,
                                   inplace = True)
```

```
reviews_sampled["reviews"].replace(to_replace=r':=\)',
                                   value='Happy', regex=True,
                                   inplace = True)
reviews_sampled["reviews"].replace(to_replace=r';=\)',
                                   value='Happy', regex=True,
                                   inplace = True)
reviews_sampled["reviews"].replace(to_replace=r':^\)',
                                   value='Happy', regex=True,
                                   inplace = True)
reviews_sampled["reviews"].replace(to_replace=r':d\)',
                                   value='Happy', regex=True,
                                   inplace = True)
reviews_sampled["reviews"].replace(to_replace=r':0\)',
                                   value='Happy', regex=True,
                                   inplace = True)
reviews_sampled["reviews"].replace(to_replace=r':~\)',
                                   value='Happy', regex=True,
                                   inplace = True)
reviews_sampled["reviews"].replace(to_replace=r';*\)',
                                   value='Happy', regex=True,
                                   inplace = True)
```

Expand contractions

Contractions python library is used to remove contractions. Along with this, manually some contractions are removed based on regex pattern matching. Example -

don't -> do not
didn't -> did not

In [14]:
```
reviews_sampled['reviews'] = reviews_sampled['reviews'].apply(
    lambda x: contractions.fix(x))
def remove_contractions(phrase):
    phrase = re.sub(r"won\'t", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
reviews_sampled['reviews'] = reviews_sampled['reviews'].apply(
    remove_contractions)
```

Remove non alphabetical characters

In [15]:
```
reviews_sampled['reviews'] = reviews_sampled['reviews'].str.replace(
    '[^a-zA-Z ]', ' ')
reviews_sampled['reviews'] = reviews_sampled['reviews'].str.strip()
reviews_sampled['reviews'] = reviews_sampled['reviews'].replace(
    r'\s+', ' ', regex=True)
```

Spell Correction

Some words that have extra letters are first fixed. Example -
Happpppppyyyyy -> Happy
Amaaazzzzing -> Amazing

Finally, Textblob python package is used on all tokens for spell correction. Example -
hapy -> Happy

In [16]:
```
rx = re.compile(r'([^\W\d_])\1{2,}')

def word_correction(text):
    correct = re.sub(r'[^\W\d_]+',
                     lambda x: Word(
                         rx.sub(r'\1\1', x.group())).correct()
                     if rx.search(
                         x.group()) else x.group(), text)
    return correct

reviews_sampled['reviews'] = reviews_sampled['reviews'].apply(
    word_correction)
```

Average length of Reviews after Data Cleaning

In [17]:
```
reviews_sampled['review_length']  = reviews_sampled['reviews'].str.len()
average_length_after_dc = reviews_sampled["review_length"].mean()
print("Average Length of Review after Data Cleaning: "
      + str(average_length_after_dc))
reviews_sampled.drop(columns=['review_length'], inplace = True)
```

```
Average Length of Review after Data Cleaning: 170.99218054866907
```

Average length of Reviews before and after Data Cleaning:

In [18]:
```
print(str(average_len_before_dc) +
      "," + str(average_length_after_dc))
```

```
176.38960904441382,170.99218054866907
```

Pre-processing ¶

Remove the stop words

Stop words are removed from reviews using nltk package stopwords list for English language

In [19]:
```
processed_samples = reviews_sampled.copy()
stop_words_list = stopwords.words('english')
processed_samples['reviews'] = processed_samples['reviews'].apply(
    lambda x: ' '.join([word for word in x.split()
                        if word not in (stop_words_list)]))
```

Drop reviews with 0 word count

Removal of stop words can lead to review character length / word count reduce to 0. These reviews are dropped.

In [20]:
```
def getReviewWordCount(text):
    return len(text.split())

processed_samples['review_word_count'] = np.array(
    processed_samples['reviews'].apply(
    getReviewWordCount))
print("Number of reviews with review length 0 : " +
      str(len(
          processed_samples[processed_samples['review_word_count'] == 0]
      )))
```

```
Number of reviews with review length 0 : 1752
```

In [21]:
```
processed_samples.drop(processed_samples[
    processed_samples['review_word_count'] == 0].index,
                           inplace = True)
```

We select 20000 reviews randomly from each rating class.

In [22]:
```
testsample = processed_samples.copy()
```

In [23]:
```
sample_count = 20000
testsample = testsample.groupby('star_rating').apply(
    lambda x: x.sample(
        n=sample_count,
        random_state=12)).reset_index(drop = True)
testsample.groupby(['star_rating'])['star_rating'].count()
```

Out[23]:
```
star_rating
1    20000
2    20000
3    20000
4    20000
5    20000
Name: star_rating, dtype: int64
```

Perform lemmatization

Lemmatization is performed after pos tagging the words. Pos tags used are -

1. J - Adjective
2. V - Verb
3. N - Noun
4. R - Adverb

In [24]:
```
def get_pos_tag(word):
    pos_tag = nltk.pos_tag([word])[0][1][0].upper()
    pos_tag_mapping = {
        "V": wordnet.VERB, "N": wordnet.NOUN,
        "J": wordnet.ADJ,  "R": wordnet.ADV
    }
    return pos_tag_mapping.get(pos_tag, wordnet.NOUN)

lemmatizer = WordNetLemmatizer()
def lemmatize_text(text):
    words = [lemmatizer.lemmatize(w, get_pos_tag(w))
             for w in nltk.word_tokenize(text)]
    return " ".join(words)

testsample['reviews'] = testsample['reviews'].apply(lemmatize_text)
```

Average length of Reviews after Data Preprocessing

In [25]:
```
testsample['review_length']  = testsample['reviews'].str.len()
average_length_after_pp = testsample["review_length"].mean()
print("Average Length of Review after Data Preprocessing: "
      + str(average_length_after_pp))
testsample.drop(columns=['review_length'], inplace = True)
```

```
Average Length of Review after Data Preprocessing: 106.50036
```

Average length of Reviews before and after Data Preprocessing:

In [26]:
```
print(str(average_length_after_dc)
      + "," + str(average_length_after_pp))
```

```
170.99218054866907,106.50036
```

TF-IDF Feature Extraction

TfidfVectorizer from sklearn python package is used. Some of the paramters set are -

1. analyzer : Set as word which specifies that each word is a token

2. ngram_range : Set as (1,3). This creates token of length 1, 2 and 3 also called as unigrams, bigrams and trigrams
3. max_df : 0.75 specifies that any word thats present in 75% of the reviews should be ignored as they do not carry any useful information
4. min_df : 2 specifies that any word thats present in lower that 2% of the reviews should be ignored as they do not carry any useful information
5. max_features : Maximum number of features to retain. This is decided based on term frequency value of the words

In [27]:
```
def tfidf_feat(max_features):
    tfidfvec = TfidfVectorizer(analyzer='word',
                               ngram_range = (1,3),
                               max_df=0.75, min_df=2,
                               max_features = max_features)
    features = tfidfvec.fit_transform(testsample['reviews'])
    return features
```

Train Test Split

We split data into 80% train and 20% test

In [28]:
```
def data_split(max_features):
    X = tfidf_feat(max_features)
    y = testsample["star_rating"].astype('int').to_numpy()
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42,
        stratify = list(testsample["star_rating"]))
    return X_train, X_test, y_train, y_test
```

Perceptron

Perceptron function from sklearn python package is used.

max_features is set as 8000 as this gave comparatively better results. The scores dropped when the number of features used were increased further

MaxAbsScaler is used. This works well with sparse matrices. This estimator scales and translates each feature individually such that the maximum absolute value of each feature in the training set will be 1.0.

In [29]:
```
X_train, X_test, y_train, y_test = data_split(8000)
perceptron_clf = make_pipeline(MaxAbsScaler(),
                               Perceptron(random_state=42,
                                          tol=1e-5,
                                          class_weight = 'balanced'))
perceptron_clf.fit(X_train, y_train)
per_score = perceptron_clf.score(X_test, y_test)

y_test_pred = perceptron_clf.predict(X_test)

report = classification_report(y_test, y_test_pred)
print(report)
report = classification_report(y_test,
                               y_test_pred,
                               output_dict = True)

class_1_precision = report["1"]["precision"]
class_1_recall = report["1"]["recall"]
class_1_f1 = report["1"]["f1-score"]
class_2_precision = report["2"]["precision"]
class_2_recall = report["2"]["recall"]
class_2_f1 = report["2"]["f1-score"]
class_3_precision = report["3"]["precision"]
class_3_recall = report["3"]["recall"]
class_3_f1 = report["3"]["f1-score"]
class_4_precision = report["4"]["precision"]
class_4_recall = report["4"]["recall"]
class_4_f1 = report["4"]["f1-score"]
class_5_precision = report["5"]["precision"]
class_5_recall = report["5"]["recall"]
class_5_f1 = report["5"]["f1-score"]
average_precision = report["macro avg"]["precision"]
average_recall = report["macro avg"]["recall"]
average_f1 = report["macro avg"]["f1-score"]

print(str(class_1_precision) + ","
      + str(class_1_recall) + "," + str(class_1_f1))
print(str(class_2_precision) + ","
      + str(class_2_recall) + "," + str(class_2_f1))
print(str(class_3_precision) + ","
      + str(class_3_recall) + "," + str(class_3_f1))
print(str(class_4_precision) + ","
      + str(class_4_recall) + "," + str(class_4_f1))
print(str(class_5_precision) + ","
      + str(class_5_recall) + "," + str(class_5_f1))
print(str(average_precision) + ","
      + str(average_recall) + "," + str(average_f1))
```

```
              precision    recall  f1-score   support

           1       0.51      0.56      0.53      4000
           2       0.35      0.31      0.33      4000
           3       0.33      0.35      0.34      4000
           4       0.37      0.36      0.37      4000
           5       0.55      0.55      0.55      4000
```

```
    accuracy                           0.43     20000
   macro avg       0.42      0.43      0.42     20000
weighted avg       0.42      0.43      0.42     20000

0.5066726984845058,0.56,0.5320033250207814
0.35339861751152074,0.30675,0.3284261241970022
0.33294117647058824,0.35375,0.343030303030303
0.37220652453120984,0.36225,0.36716077537058156
0.5509586276488395,0.546,0.5484681064791562
0.4232355289293328,0.42575,0.4238177268195649
```

SVM

LinearSVC function from sklearn python package is used.

max_features is set as 2000 as this gave comparatively better results. The scores dropped when the number of features used were increased further

In [30]:
```
X_train, X_test, y_train, y_test = data_split(2000)
svm_linear_clf = LinearSVC(random_state=12, tol=1e-5,
                           max_iter = 20000, class_weight = 'balanced')
svm_linear_clf.fit(X_train, y_train)
svm_score = svm_linear_clf.score(X_test, y_test)

y_test_pred = svm_linear_clf.predict(X_test)

report = classification_report(y_test, y_test_pred)
print(report)
report = classification_report(y_test, y_test_pred, output_dict = True)

class_1_precision = report["1"]["precision"]
class_1_recall = report["1"]["recall"]
class_1_f1 = report["1"]["f1-score"]
class_2_precision = report["2"]["precision"]
class_2_recall = report["2"]["recall"]
class_2_f1 = report["2"]["f1-score"]
class_3_precision = report["3"]["precision"]
class_3_recall = report["3"]["recall"]
class_3_f1 = report["3"]["f1-score"]
class_4_precision = report["4"]["precision"]
class_4_recall = report["4"]["recall"]
class_4_f1 = report["4"]["f1-score"]
class_5_precision = report["5"]["precision"]
class_5_recall = report["5"]["recall"]
class_5_f1 = report["5"]["f1-score"]
average_precision = report["macro avg"]["precision"]
average_recall = report["macro avg"]["recall"]
average_f1 = report["macro avg"]["f1-score"]

print(str(class_1_precision) + ","
      + str(class_1_recall) + "," + str(class_1_f1))
print(str(class_2_precision) + ","
      + str(class_2_recall) + "," + str(class_2_f1))
print(str(class_3_precision) + ","
      + str(class_3_recall) + "," + str(class_3_f1))
print(str(class_4_precision) + ","
      + str(class_4_recall) + "," + str(class_4_f1))
print(str(class_5_precision) + ","
      + str(class_5_recall) + "," + str(class_5_f1))
print(str(average_precision) + ","
      + str(average_recall) + "," + str(average_f1))
```

```
              precision    recall  f1-score   support

           1       0.55      0.67      0.60      4000
           2       0.40      0.34      0.37      4000
           3       0.42      0.33      0.37      4000
           4       0.46      0.42      0.44      4000
           5       0.61      0.74      0.67      4000

    accuracy                           0.50     20000
   macro avg       0.49      0.50      0.49     20000
weighted avg       0.49      0.50      0.49     20000
```

```
0.5458966565349544,0.6735,0.6030218242865137
0.4035767511177347,0.3385,0.3681849082256968
0.4210028382213813,0.33375,0.37233300794868224
0.45905231443440153,0.419,0.4381126650111097
0.6057692307692307,0.74025,0.6662916291629163
0.48705955821554053,0.501,0.4895888069269837
```

Logistic Regression

LogisticRegression function from sklearn python package is used.

max_features is set as 20000 as this gave comparatively better results. The scores dropped when the number of features used were increased further

In [31]:
```
X_train, X_test, y_train, y_test = data_split(20000)
logistic_regression_clf = LogisticRegression(max_iter=30000,
                                             tol=1e-5,
                                             random_state=42,
                                             solver='saga',
                                             class_weight = 'balanced')
logistic_regression_clf.fit(X_train, y_train)
log_score = logistic_regression_clf.score(X_test, y_test)

y_test_pred = logistic_regression_clf.predict(X_test)

report = classification_report(y_test, y_test_pred)
print(report)
report = classification_report(y_test, y_test_pred, output_dict = True)

class_1_precision = report["1"]["precision"]
class_1_recall = report["1"]["recall"]
class_1_f1 = report["1"]["f1-score"]
class_2_precision = report["2"]["precision"]
class_2_recall = report["2"]["recall"]
class_2_f1 = report["2"]["f1-score"]
class_3_precision = report["3"]["precision"]
class_3_recall = report["3"]["recall"]
class_3_f1 = report["3"]["f1-score"]
class_4_precision = report["4"]["precision"]
class_4_recall = report["4"]["recall"]
class_4_f1 = report["4"]["f1-score"]
class_5_precision = report["5"]["precision"]
class_5_recall = report["5"]["recall"]
class_5_f1 = report["5"]["f1-score"]
average_precision = report["macro avg"]["precision"]
average_recall = report["macro avg"]["recall"]
average_f1 = report["macro avg"]["f1-score"]

print(str(class_1_precision) + ","
      + str(class_1_recall) + "," + str(class_1_f1))
print(str(class_2_precision) + ","
      + str(class_2_recall) + "," + str(class_2_f1))
print(str(class_3_precision) + ","
      + str(class_3_recall) + "," + str(class_3_f1))
print(str(class_4_precision) + ","
      + str(class_4_recall) + "," + str(class_4_f1))
print(str(class_5_precision) + ","
      + str(class_5_recall) + "," + str(class_5_f1))
print(str(average_precision) + ","
      + str(average_recall) + "," + str(average_f1))
```

```
              precision    recall  f1-score   support

           1       0.59      0.63      0.61      4000
           2       0.41      0.39      0.40      4000
           3       0.41      0.39      0.40      4000
           4       0.48      0.45      0.46      4000
           5       0.65      0.70      0.67      4000

    accuracy                           0.51     20000
```

```
   macro avg       0.51      0.51      0.51     20000
weighted avg       0.51      0.51      0.51     20000

0.5863109048723898,0.63175,0.6081829121540313
0.4105235738473561,0.394,0.40209210358464087
0.41315300981172104,0.3895,0.40097799511002447
0.4790450928381963,0.4515,0.4648648648648649
0.6503480278422273,0.70075,0.6746089049338146
0.5078761218423782,0.5135,0.5101453561294752
```

Naive Bayes

MultinomialNB function from sklearn python package is used.

max_features is set as 20000 as this gave comparatively better results. The scores dropped when the number of features used were increased further

In [32]:
```
X_train, X_test, y_train, y_test = data_split(20000)
mnb_clf = MultinomialNB()
mnb_clf.fit(X_train, y_train)
naive_score = mnb_clf.score(X_test, y_test)

y_test_pred = mnb_clf.predict(X_test)

report = classification_report(y_test, y_test_pred)
print(report)
report = classification_report(y_test, y_test_pred, output_dict = True)

class_1_precision = report["1"]["precision"]
class_1_recall = report["1"]["recall"]
class_1_f1 = report["1"]["f1-score"]
class_2_precision = report["2"]["precision"]
class_2_recall = report["2"]["recall"]
class_2_f1 = report["2"]["f1-score"]
class_3_precision = report["3"]["precision"]
class_3_recall = report["3"]["recall"]
class_3_f1 = report["3"]["f1-score"]
class_4_precision = report["4"]["precision"]
class_4_recall = report["4"]["recall"]
class_4_f1 = report["4"]["f1-score"]
class_5_precision = report["5"]["precision"]
class_5_recall = report["5"]["recall"]
class_5_f1 = report["5"]["f1-score"]
average_precision = report["macro avg"]["precision"]
average_recall = report["macro avg"]["recall"]
average_f1 = report["macro avg"]["f1-score"]

print(str(class_1_precision) + ","
      + str(class_1_recall) + "," + str(class_1_f1))
print(str(class_2_precision) + ","
      + str(class_2_recall) + "," + str(class_2_f1))
print(str(class_3_precision) + ","
      + str(class_3_recall) + "," + str(class_3_f1))
print(str(class_4_precision) + ","
      + str(class_4_recall) + "," + str(class_4_f1))
print(str(class_5_precision) + ","
      + str(class_5_recall) + "," + str(class_5_f1))
print(str(average_precision) + ","
      + str(average_recall) + "," + str(average_f1))
```

```
              precision    recall  f1-score   support

           1       0.58      0.65      0.61      4000
           2       0.41      0.35      0.38      4000
           3       0.41      0.40      0.41      4000
           4       0.47      0.43      0.45      4000
           5       0.63      0.72      0.67      4000

    accuracy                           0.51     20000
   macro avg       0.50      0.51      0.50     20000
weighted avg       0.50      0.51      0.50     20000

0.5764705882352941,0.64925,0.6106995884773662
```

```
0.4145550972304066,0.35175,0.380578847714363
0.4121552604698672,0.4035,0.40778170793329965
0.4691527282698108,0.42775,0.44749574996730745
0.6342000881445571,0.7195,0.6741625673459827
0.5013067524699872,0.5103500000000001,0.5041436922876638
```