Fundamentals of Data Analytics and Learning

HW1 – First Visit in Kaggle Data

3/9/2022

Name: Hoe Zi Onn / 何子安

Student ID: E44065020

Year: PSY 110 / 心理110

Table of Contents

[Data Description 1](#_Toc97760782)

[Take a Look at the Dataset 1](#_Toc97760783)

[EDA 2](#_Toc97760784)

[Labels Distribution 2](#_Toc97760785)

[Apply label encoding into Sentiment column 3](#_Toc97760786)

[Wordcloud of Sentence’s Text 4](#_Toc97760787)

[Cleaning the Sentences 5](#_Toc97760788)

[X-y Preparation 5](#_Toc97760789)

[Model Training and Model Evaluation 6](#_Toc97760790)

[Naive Bayes 6](#_Toc97760791)

[X-y Preparation (using TF-IDF) 6](#_Toc97760792)

[Model Training and Model Evaluation (using TF-IDF) 7](#_Toc97760793)

[Naive Bayes 7](#_Toc97760794)

[MultinominalNB 8](#_Toc97760795)

[CatBoost 8](#_Toc97760796)

[XGBoost 8](#_Toc97760797)

[Comparision Between Difference Notebook’s Code 9](#_Toc97760798)

[Thoughts on This Dataset Among Others 9](#_Toc97760799)

[References 10](#_Toc97760800)

# Data Description

Data Source:  
- <https://www.kaggle.com/sbhatti/financial-sentiment-analysis>

Dataset:  
- Financial Sentiment Analysis (Financial sentences with sentiment labels)

Introduction of Dataset:  
- The following data is intended for advancing financial sentiment analysis research. It’s two datasets (FiQA, Financial PhraseBank) combined into one easy-to-use CSV file. It provides financial sentences with sentiment labels.

Citations:  
- Malo, Pekka, et al. “Good debt or bad debt: Detecting semantic orientations in economic texts.” Journal of the Association for Information Science and Technology 65.4 (2014): 782-796.

Target:  
- Classification

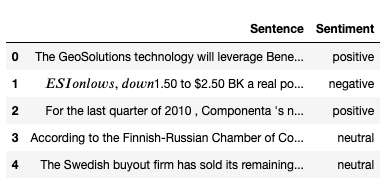
This dataset contains only single file and saved in *.csv* fomat. We will review the following sections through *python notebook cell* with theirs outputs. And all the libraries/packages code will be skipped. (Complete code are attached to this report.)

## Take a Look at the Dataset

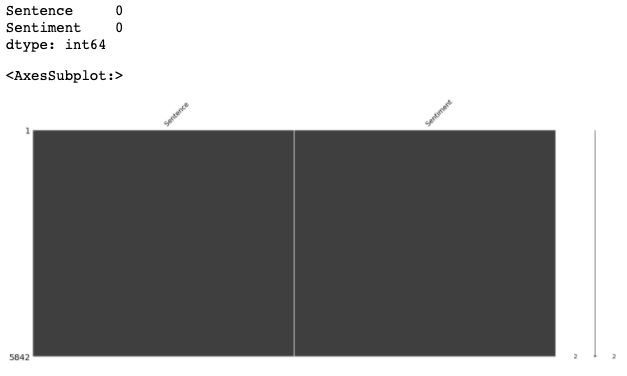
This dataset has only two columns which is Sentence and Sentiment.

Sentence: Sentences that we are going to classify and analysis. (features)  
Sentiment: Answers that we are going to predict. (labels)

df = pd.read\_csv('data.csv')  
df.head()



print(df.isnull().sum())  
msn.matrix(df)



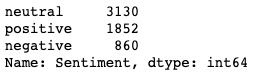
* There is **no missing value** in the dataset.

# EDA

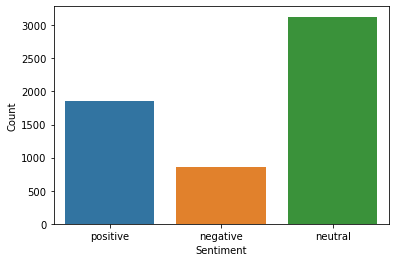
## Labels Distribution

Counting the numbers of labels.

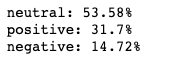
df['Sentiment'].value\_counts()



sns.countplot(x='Sentiment', data=df)  
plt.xlabel('Sentiment')  
plt.ylabel('Count')  
plt.show()



neutral, pos, neg = df['Sentiment'].value\_counts()  
total = neutral + pos + neg  
for i, j in zip(['neutral', 'positive', 'negative'], df['Sentiment'].value\_counts()):  
 print(f'{i}: {round(j/total\*100, 2)}%')



* These values indicate the dataset is **imbalanced**.

# Apply label encoding into Sentiment column

The labels of Sentiment column need to be translate into numbers.

**Label Encoding** refers to converting the labels into a numeric form so as to convert them into the machine-readable form. Machine learning algorithms can then decide in a better way how those labels must be operated.

label\_encoder = preprocessing.LabelEncoder()  
df['Sentiment'] = label\_encoder.fit\_transform(df['Sentiment'])  
df['Sentiment'].unique()  
  
df.head()



# Wordcloud of Sentence’s Text

By putting all words together with *stopwords* being removed, we shall see the word that showed up more often will be bigger than those who barely showed.

stopwords\_ = set(STOPWORDS)  
  
text = " ".join(i for i in df.Sentence)  
wordcloud = WordCloud(stopwords=stopwords\_, background\_color="white").generate(text)  
  
plt.imshow(wordcloud, interpolation='bilinear')  
plt.axis("off")  
plt.show()

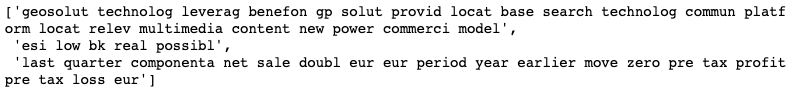


# Cleaning the Sentences

Real-world datasets often contains a lot of incorrect/incomplete informations. I onces heard that your insight and analysis are only as good as the data you are using. Also as garbage data in, garbage analysis out. So in this step, we are going to ‘clean’ our data like removing stopwords, standardize the data and so on.

**Data cleaning** is the process of fixing or removing incorrect, corrupted, incorrectly formatted, duplicate, or incomplete data within a dataset. When combining multiple data sources, there are many opportunities for data to be duplicated or mislabeled. If data is incorrect, outcomes and algorithms are unreliable, even though they may look correct.

nltk.download('stopwords')  
  
corpus = []  
for i in range(0, len(df)):  
 review = re.sub('[^a-zA-Z]', ' ', df['Sentence'][i])  
 review = review.lower()  
 review = review.split()  
 ps = PorterStemmer()  
 all\_stopwords = stopwords.words('english')  
 all\_stopwords.remove('not')  
 review = [ps.stem(word)   
 for word in review   
 if not word in set(all\_stopwords)]  
 review = ' '.join(review)  
 corpus.append(review)  
  
corpus[0: 3]



# 

# X-y Preparation

For most ML algorithms are required x (features) , y (labels) as the inputs and we need to vectorize the features before we give it to our models, called **features extraction**.

cv = CountVectorizer(max\_features=1500, ngram\_range=(1,3))  
X = cv.fit\_transform(corpus).toarray()  
y = df.iloc[:, -1].values

## Model Training and Model Evaluation

After splitting our data into *training data* and *test data*, we use *training data* to train our models and *test data* to evaluate our model’s performance.

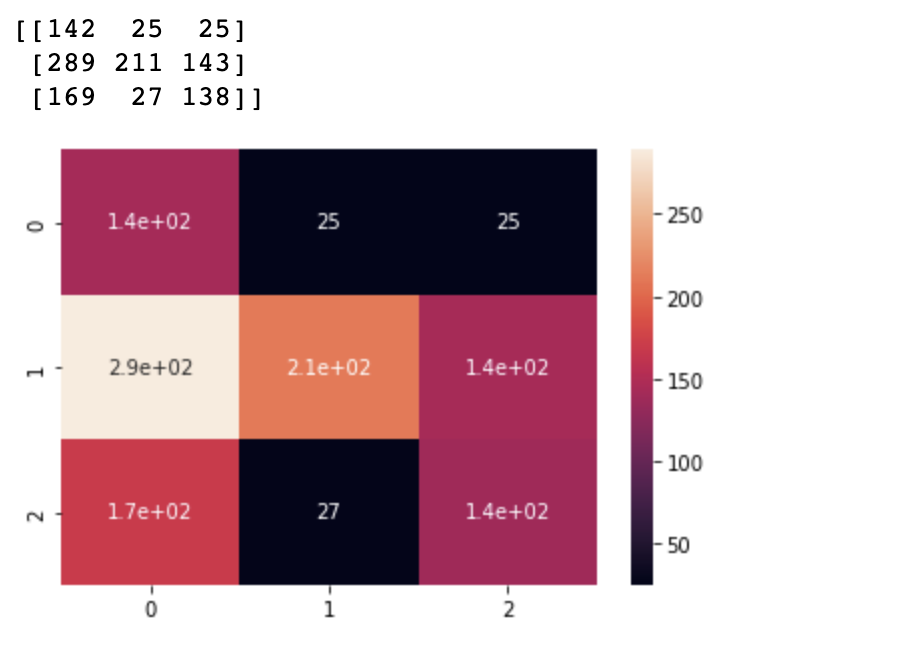
I ran only Naive Bayes in this part, and the result will be the **baseline** to our improved method.

*metrics*: confusion\_matrix, accuracy\_score

**Confusion matrix**: a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class. The confusion matrix shows the ways in which your classification model is confused when it makes predictions.

### Naive Bayes

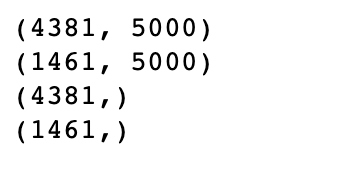
classifier = GaussianNB()  
classifier.fit(X\_train, y\_train)  
y\_pred = classifier.predict(X\_test)  
  
cm = confusion\_matrix(y\_test, y\_pred)  
print(cm)  
  
sns.heatmap(cm, annot=True)  
plt.show()



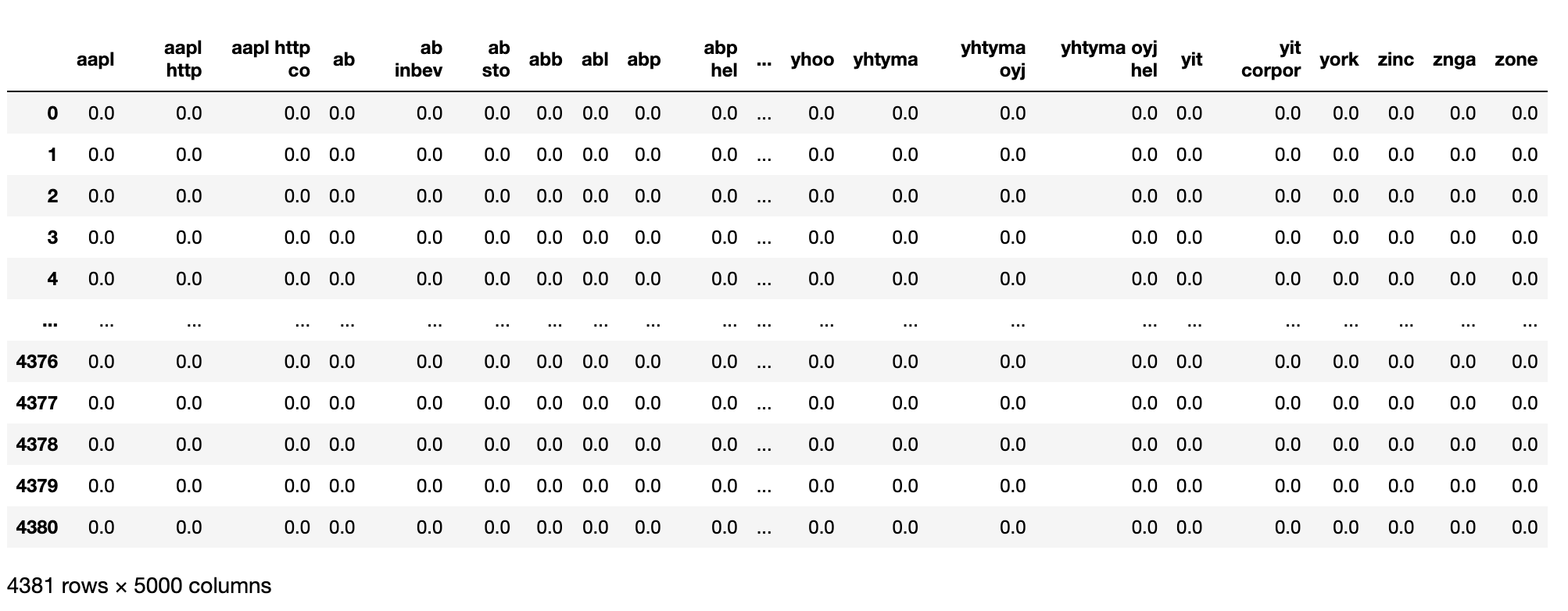
# X-y Preparation (using TF-IDF)

**TF-IDF** (term frequency-inverse document frequency) can quantify the importance or relevance of string representations (words, phrases, lemmas, etc) in a document amongst a collection of documents (also known as a corpus).

tfidf\_v = TfidfVectorizer(max\_features=5000, ngram\_range=(1,3))  
X = tfidf\_v.fit\_transform(corpus).toarray()  
y = df['Sentiment']  
  
X1\_train, X1\_test, y1\_train, y1\_test = train\_test\_split(  
 X, y, test\_size = 0.25,random\_state = 0)  
  
print(X1\_train.shape)  
print(X1\_test.shape)  
print(y1\_train.shape)  
print(y1\_test.shape)



count\_df = pd.DataFrame(X1\_train, columns=tfidf\_v.get\_feature\_names())  
count\_df



## 

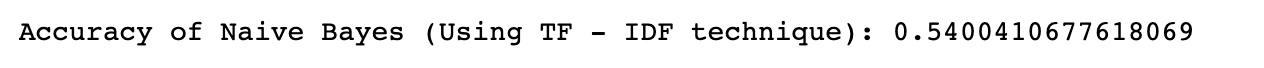
## Model Training and Model Evaluation (using TF-IDF)

I ran multiple models (Naive Bayes, MultinominalNB, CatBoost, XGBoost) in this part as to be compare to other models.

*metrics*: confusion\_matrix, accuracy\_score

### Naive Bayes

classifier = GaussianNB()  
classifier.fit(X1\_train, y1\_train)  
y1\_pred = classifier.predict(X1\_test)  
  
acc2 = accuracy\_score(y1\_test, y1\_pred)  
print(f"Accuracy of Naive Bayes (Using TF - IDF technique): {acc2}")



### MultinominalNB

classifier = MultinomialNB()  
classifier.fit(X1\_train, y1\_train)  
pred = classifier.predict(X1\_test)  
score = accuracy\_score(y1\_test, pred)  
score

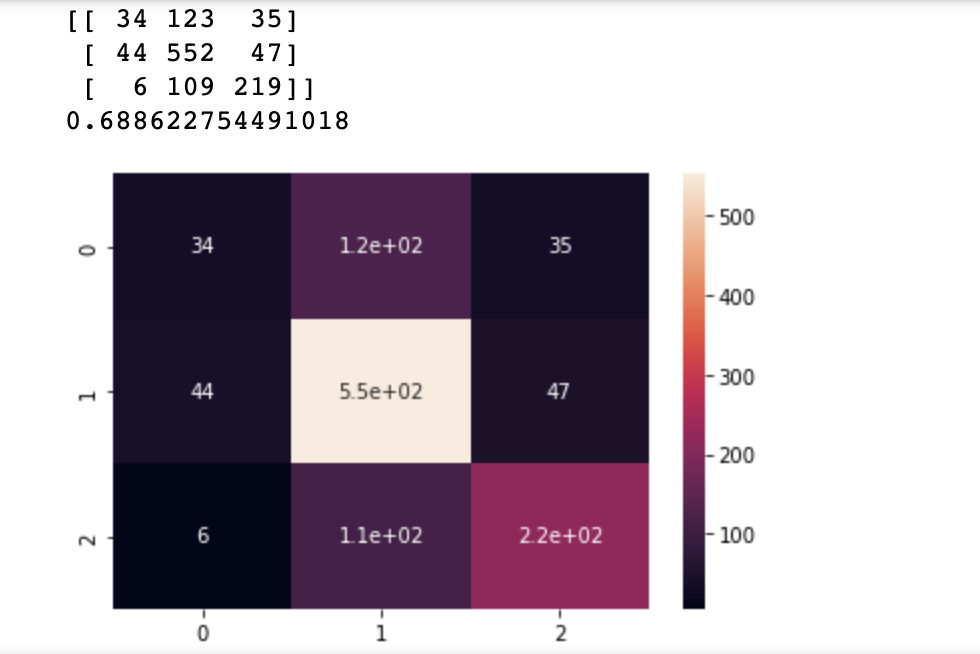


### 

### CatBoost

**CatBoost** is an algorithm for gradient boosting on decision trees.

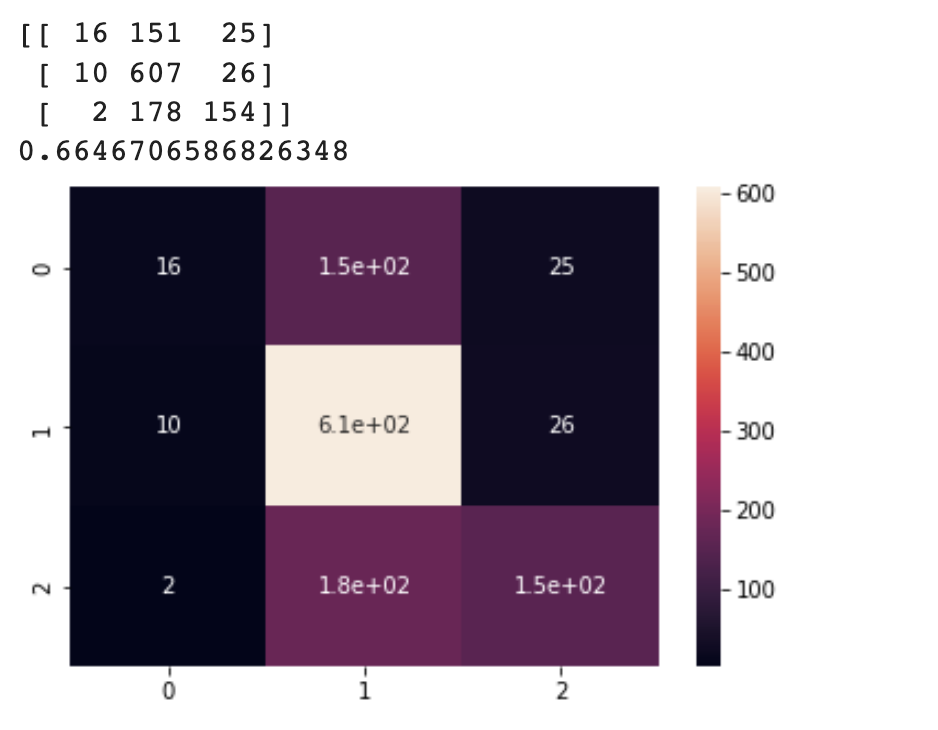
classifier = CatBoostClassifier()  
classifier.fit(X\_train, y\_train)  
y\_pred = classifier.predict(X\_test)  
  
cm = confusion\_matrix(y\_test, y\_pred)  
print(cm)  
  
acc4 = accuracy\_score(y\_test, y\_pred)  
print(acc4)  
  
sns.heatmap(cm,annot=True)  
plt.show()



### XGBoost

**XGBoost** is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way.

classifier = XGBClassifier()  
classifier.fit(X\_train, y\_train)  
y\_pred = classifier.predict(X\_test)  
  
cm = confusion\_matrix(y\_test, y\_pred)  
print(cm)  
  
acc3 = accuracy\_score(y\_test, y\_pred)  
print(acc3)  
  
sns.heatmap(cm, annot=True)  
plt.show()



# Comparision Between Difference Notebook’s Code

1. Different *packages* and *ML models* are being used.
2. Some even apply different *K-fold* method.
3. Different *evaluation metrics* are being used.
4. Different *text processing* would lead to different results on varies ML models.

# Thoughts on This Dataset Among Others

* This dataset is not like the other classical’s dataset. It’s a **NLP (Natural Language Processing)** task, aims to do **classification**.
* There are lot of way to do text processing in NLP task, and I only scratch the surface.

# References

1. [Data Cleaning](https://www.tableau.com/learn/articles/what-is-data-cleaning)
2. [Label Encoding](https://www.geeksforgeeks.org/ml-label-encoding-of-datasets-in-python/#:~:text=Label%20Encoding%20refers%20to%20converting,structured%20dataset%20in%20supervised%20learning.)
3. [TF-IDF](https://www.capitalone.com/tech/machine-learning/understanding-tf-idf/)
4. [Confusion Matrix](https://machinelearningmastery.com/confusion-matrix-machine-learning/#:~:text=A%20confusion%20matrix%20is%20a%20summary%20of%20prediction%20results%20on,key%20to%20the%20confusion%20matrix.)
5. [CatBoost](https://catboost.ai/)
6. [XGBoost](https://xgboost.readthedocs.io/en/stable/)
7. [Kaggle - Dataset](https://www.kaggle.com/sbhatti/financial-sentiment-analysis)
8. [Kaggle - Code](https://www.kaggle.com/sanjoymondal0/financial-sentiment-analysis)