nana direct

APP REVIEWS ANALYSIS

Predict if The User is Satisfied with The App or Not

Abdulrahman AbouOuf

Nana Internship - 2020

Supervisor

Muhammed Mustafa

Table of Contents

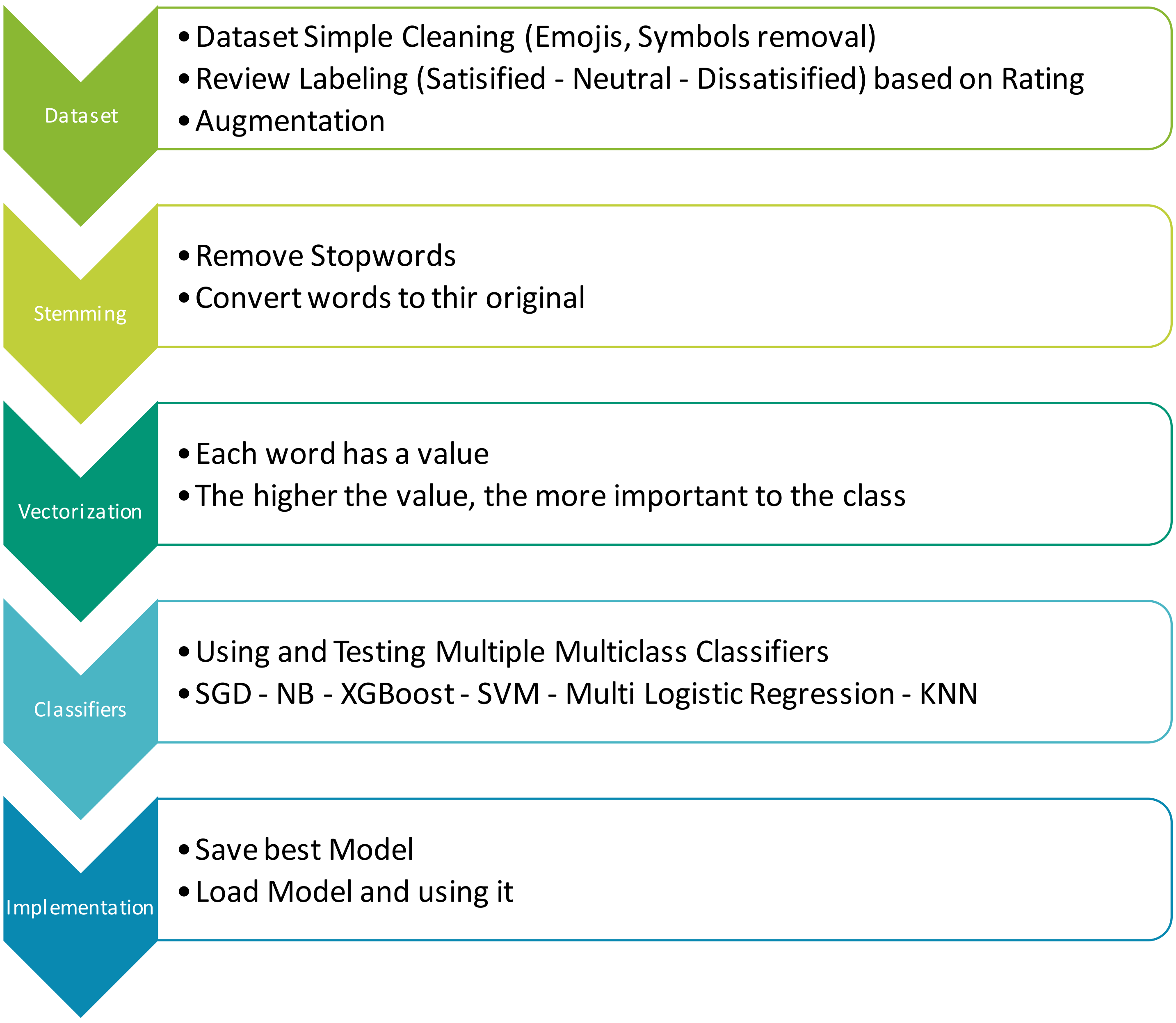
1 Introduction

The NLP system proposed is used to determine if the App user (Android – iOS) is satisfied or not based on his/her text review. Using NLP text and Sentiment Analysis to retrieve information from these text reviews.

1.1 Problem Statement

Using Text Classification and prober Data Cleaning to build a machine learning model to predict if the user is satisfied or not.

1.2 System Architecture


Dataset
• Dataset Simple Cleaning (Emojis, Symbols removal)
• Review Labeling (Satisified - Neutral - Dissatisified) based on Rating
• Augmentation
Stemming
• Remove Stopwords
• Convert words to thir original
Vectorization
• Each word has a value
• The higher the value, the more important to the class
Classifiers
• Using and Testing Multiple Multiclass Classifiers
• SGD - NB - XGBoost - SVM - Multi Logistic Regression - KNN
Implementation
• Save best Model
• Load Model and using it

Each of these will be discussed next in detail (Technically with codes)

2 Dataset

The provided datasets were 2:

- iOS: 2534 reviews
  - Arabic: 2086 reviews
  - English: 271 reviews
- Android: 8767 reviews
  - Arabic: 3398 reviews
  - English: 417 reviews

2.1 Simple Cleaning

Removing unnecessary symbols and emojis for the text classifier. As follow

```
def remove_emoji(string):
    emoji_pattern = re.compile("["
                               u"\U0001F600-\U0001F64F"  # emoticons
                               u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                               u"\U0001F680-\U0001F6FF"  # transport & map symbols
                               u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                               u"\U00002702-\U000027B0"
                               u"\U000024C2-\U0001F251"
                               "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r'', string)
```

2.2 Language detection

As the reviews are combination of Arabic and English, I used langdetect library as the labeling of the sentences that came with data is incorrect.

Here is how I used langdetect:

```
!pip install langdetect
```

```
Requirement already satisfied: langdetect in c:\programdata\anaconda3\lib\site-packages (1.0.8)
Requirement already satisfied: six in c:\programdata\anaconda3\lib\site-packages (from langdetect) (1.14.0)
```

```
def detect_lang(string):
    try:
        language = langdetect.detect(string)
    except:
        language = "error"
    return language
```

```
android_reviews_full['detected Language'] = android_reviews_full['Review Text'].apply(lambda x: detect_lang(x))
```

Now the data is ready to be separated to Arabic data and English data. Each language has its classifier model.

2.3 Data labeling

As the classifier has to be trained on supervised dada (Sentence – Label) I used the Star Rating as the Label

The labels are (Satisfied – Dissatisfied – Neutral). Here as follow

```
def detect_label(i):
    rate = int(i)
    if rate >= 4:
        return "satisfied"
    elif rate == 3:
        return "neutral"
    elif rate <= 2:
        return "dissatisfied"
```

2.4 Dataset Classes samples

After separating data to English and Arabic. And Classify the sentences (based on Rating) the data are as follow:

Language/Class Sentences	Satisfied	Dissatisfied	Neutral
English[1]	2565	3321	306
Arabic	2194	3051	239

2.5 Augmentation

As we can see, the English dataset is small compared to Arabic dataset. I used EDA (Easy Data Augmentation) as follow

2.5.1 EDA (Easy data Augmentation)

I used EDA (Easy Data Augmentation) as it deals better with Text Classification datasets to add more sentences to each class, hence avoiding overfitting.

2.5.1.1 How EDA Works?

Given a sentence in the training set, EDA applies the following:

- **Synonym Replacement (SR):** Randomly choose *n words* from the sentence that are not stop words. Replace each of these words with one of its synonyms chosen at random.
- **Random Insertion (RI):** Find a random synonym of a random word in the sentence that is not a stop word. Insert that synonym into a random position in the sentence. Do this *n* times.
- **Random Swap (RS):** Randomly choose two words in the sentence and swap their positions. Do this *n* times.
- **Random Deletion (RD):** For each word in the sentence, randomly remove it with *probability.*

```
pip install - U nltk
import nltk;
nltk.download('wordnet')

# Cloning the Repo
!git clone https: //github.com/jasonwei20/eda_nlp.git

#The input files in the format label\tsentence(note the\t (tab))
!python augment.py--input = 'input.txt'--output = output_augmented_16sens_0.05 alpaha.txt--num_aug = 16--alpha = 0.05
```

Now we have the dataset needed for our model.

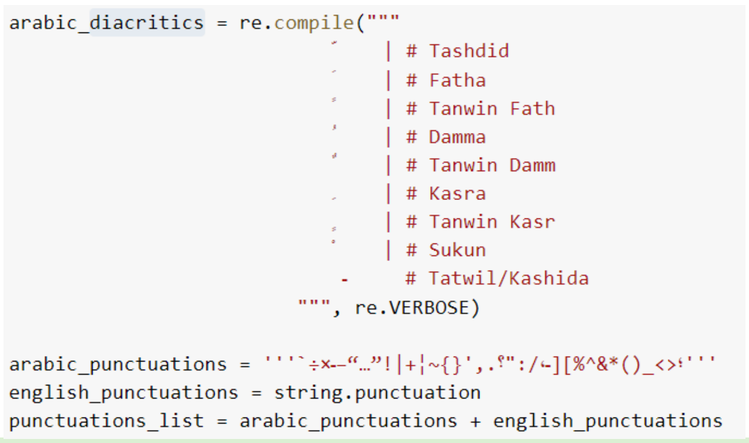
[1] We see the sample are increased using Augmentation Techniques. Will be discussed.

3 Stemming

Stemming is the process of reducing inflected words to their word stem, base or root form—generally a written word form.

So, to stem Arabic words I used many techniques as follow:

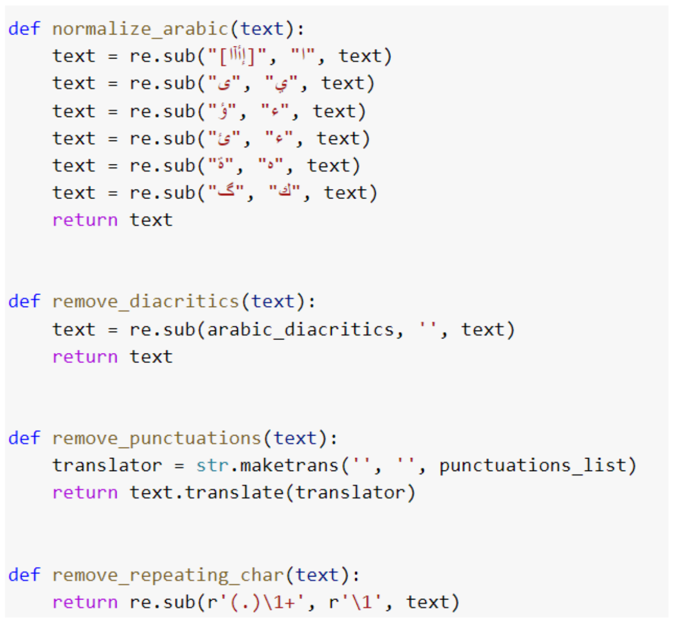
3.1 Remove Diacritics



```python
arabic_diacritics = re.compile("""
                             ّ    | # Tashdid
                             َ    | # Fatha
                             ً    | # Tanwin Fath
                             ُ    | # Damma
                             ٌ    | # Tanwin Damm
                             ِ    | # Kasra
                             ٍ    | # Tanwin Kasr
                             ْ    | # Sukun
                             ـ     # Tatwil/Kashida
                         """, re.VERBOSE)

arabic_punctuations = '''`÷×؛<>_()*&^%][ـ،/:"؟.,'{}~¦+|!”…“–ـ'''
english_punctuations = string.punctuation
punctuations_list = arabic_punctuations + english_punctuations
```

3.2 Normalization and Light Stemming

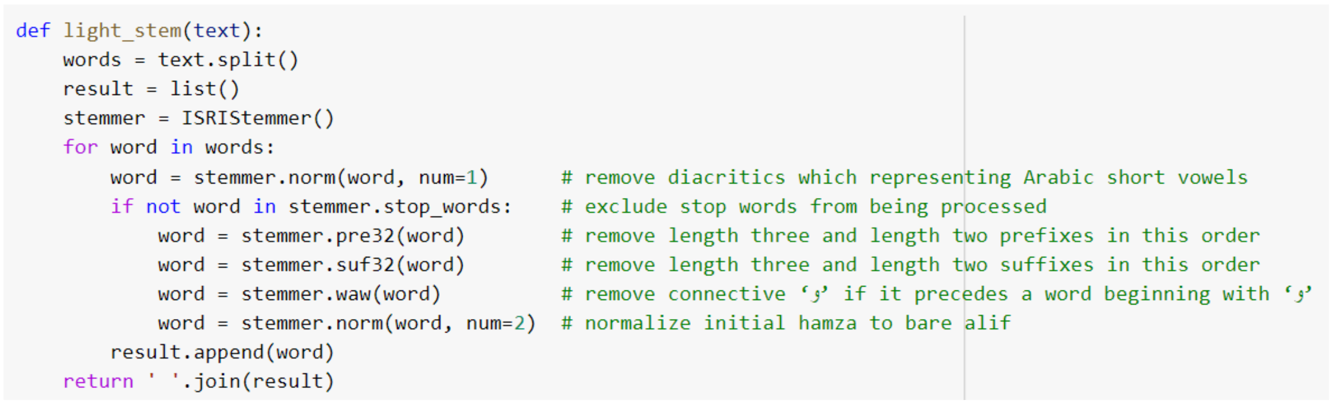


```python
def normalize_arabic(text):
    text = re.sub("[إأآا]", "ا", text)
    text = re.sub("ى", "ي", text)
    text = re.sub("ؤ", "ء", text)
    text = re.sub("ئ", "ء", text)
    text = re.sub("ة", "ه", text)
    text = re.sub("گ", "ك", text)
    return text


def remove_diacritics(text):
    text = re.sub(arabic_diacritics, '', text)
    return text


def remove_punctuations(text):
    translator = str.maketrans('', '', punctuations_list)
    return text.translate(translator)


def remove_repeating_char(text):
    return re.sub(r'(.)\1+', r'\1', text)
```



```python
def light_stem(text):
    words = text.split()
    result = list()
    stemmer = ISRIStemmer()
    for word in words:
        word = stemmer.norm(word, num=1)      # remove diacritics which representing Arabic short vowels
        if not word in stemmer.stop_words:    # exclude stop words from being processed
            word = stemmer.pre32(word)        # remove length three and length two prefixes in this order
            word = stemmer.suf32(word)        # remove length three and length two suffixes in this order
            word = stemmer.waw(word)          # remove connective ‘و’ if it precedes a word beginning with ‘و’
            word = stemmer.norm(word, num=2)  # normalize initial hamza to bare alif
        result.append(word)
    return ' '.join(result)
```

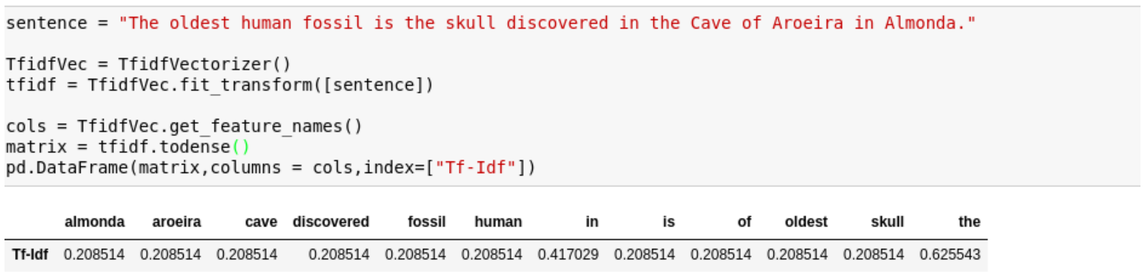
4 Vectorization and Pipeline

Text Vectorization is the process of converting text into numerical representation so that the machine can understand, so each word of our sentence will have its vector and each vector represent how important this word is. Techniques vary (Bag of Words, etc.) but BoW doesn't do good with noise and doesn't give importance to certain words that distinguish each class.

Word embedding is the collective name for a set of language modeling and feature learning techniques in natural language processing where words or phrases from the vocabulary are mapped to vectors of real numbers. So, the classifier can deal with the numbers.

4.1 TF-IDF

TF-IDF stands for "Term Frequency — Inverse Document Frequency". This is a technique to quantify a word in documents, we generally compute a weight to each word which signifies the importance of the word in the document and corpus. This method is a widely used technique in Information Retrieval and Text Mining.



```
sentence = "The oldest human fossil is the skull discovered in the Cave of Aroeira in Almonda."

TfidfVec = TfidfVectorizer()
tfidf = TfidfVec.fit_transform([sentence])

cols = TfidfVec.get_feature_names()
matrix = tfidf.todense()
pd.DataFrame(matrix,columns = cols,index=["Tf-Idf"])
```

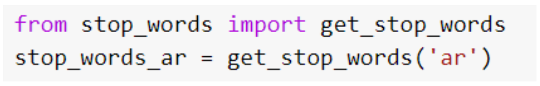
	almonda	aroeira	cave	discovered	fossil	human	in	is	of	oldest	skull	the
Tf-Idf	0.208514	0.208514	0.208514	0.208514	0.208514	0.208514	0.417029	0.208514	0.208514	0.208514	0.208514	0.625543

Convert a collection of raw documents to a matrix of TF-IDF features.

TF-IDF downscales weights for words (Vectors) that occur in many sentences (less informative) and distinguishes the important words of each class.

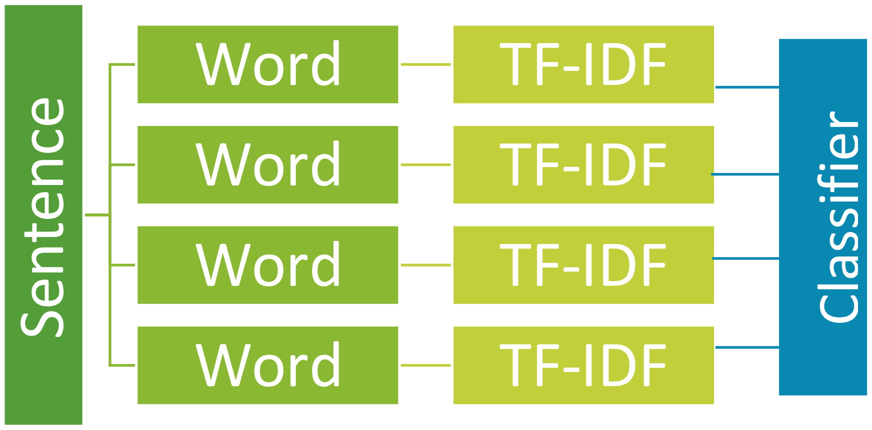
4.2 Stopwords

Removing Stopwords from Arabic and English as follow



```
from stop_words import get_stop_words
stop_words_ar = get_stop_words('ar')
```

4.3 Classifier Pipeline



5 Modeling

Here I will illustrate the training and testing data, The Classifiers used and so on

5.1 Train-Test Split

I splinted the data with (90% training – 10% testing) as the data isn't fairly huge, so I needed to add more the training data,as follow

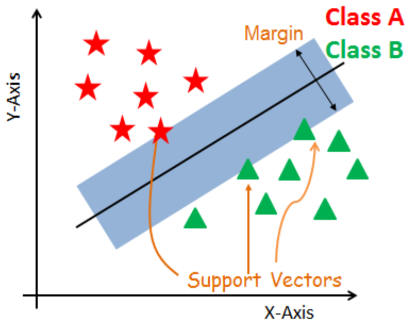
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.1, random_state = 0)
```

5.2 Classifiers

I used many classifiers and methods to achieve the highest accuracies as follow

5.2.1 SVC (Support Vector Classifier)

SVM is an exciting algorithm and the concepts are relatively simple. The classifier separates data points using a hyperplane with the largest amount of margin. That's why an SVM classifier is also known as a discriminative classifier. SVM finds an optimal hyperplane which helps in classifying new data points.

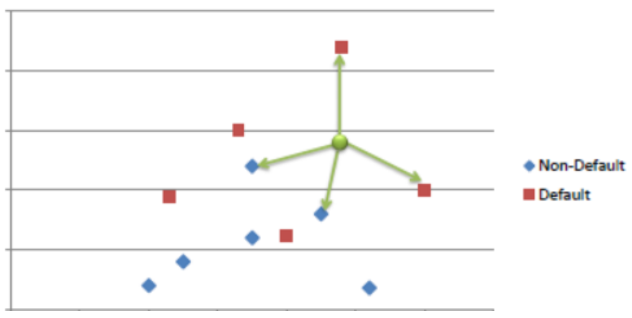


```
from sklearn.svm import SVC

clfSVC= SVC(kernel = 'rbf', random_state = 0)
clfSVC.fit(X_train_res, y_train_res)
clfSVC.score(X_train_res, y_train_res)
```

5.2.2 KNN (K Nearest Neighbor)

k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor. In our case (n=2 for 3 classifiers)



```
from sklearn.neighbors import KNeighborsClassifier
clfKNN = KNeighborsClassifier(n_neighbors = 2, metric = 'minkowski', p = 2)
clfKNN.fit(X_train_res, y_train_res)
clfKNN.score(X_train_res, y_train_res)
```

5.2.3 NB (Naïve Bayes)

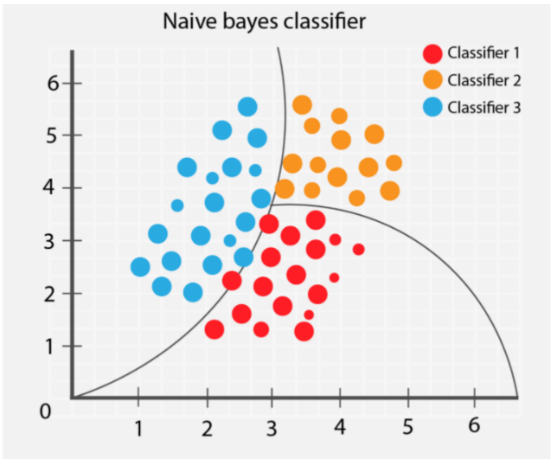
Naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naïve) independence assumptions between the features.

Naïve Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem.

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

using Bayesian probability terminology, the above equation can be written as

$$\text{Posterior} = \frac{\text{prior x likelihood}}{\text{evidence}}$$



```
from sklearn.naive_bayes import MultinomialNB

clfNB= MultinomialNB()
clfNB.fit(X_train_tfidf, y_train)
clfNB.score(X_train_tfidf, y_train)
```

5.2.4 XGBoost

XGBoost (eXtreme Gradient Boosting) is an advanced implementation of gradient boosting algorithm.

- XGBoost is a powerful machine learning algorithm especially where speed and accuracy are concerned
- We need to consider different parameters and their values to be specified while implementing an XGBoost model
- The XGBoost model requires parameter tuning to improve and fully leverage its advantages over other algorithms

eXtreme Gradient Boosting
custom tree building algorithm
XGBoost
by Tianqi Chen
Used for: • classification • regression • ranking with custom loss functions
Interfaces for Python and R, can be executed on YARN

```
X_test_tfidf=count_vect.transform(X_test)
D_train = xgb.DMatrix(X_train_res, label=y_train_res)
D_test = xgb.DMatrix(X_test_tfidf, label=y_test)
```

```
parameters = {
     "eta"    : [0.05, 0.10, 0.15, 0.20, 0.25, 0.30 ] ,
     "max_depth"        : [ 3, 4, 5, 6, 8, 10, 12, 15],
     "min_child_weight" : [ 1, 3, 5, 7 ],
     "gamma"            : [ 0.0, 0.1, 0.2 , 0.3, 0.4 ],
     "colsample_bytree" : [ 0.3, 0.4, 0.5 , 0.7 ]
     }
```

For parameter tuning visit link, link and link.

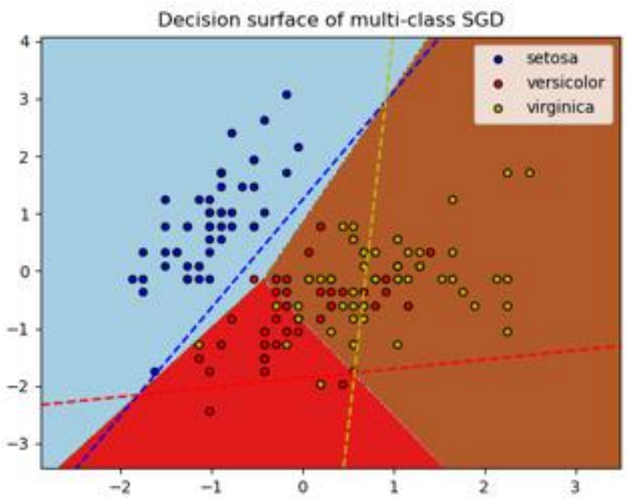
5.2.5 SGD (Stochastic Gradient Descent) Classifier

Linear classifiers (SVM, logistic regression, etc.) with SGD training.

This is also a linear model of classification but with stochastic gradient descent learning so that the loss is estimated each sample at a time and the model is updated along the way.

This estimator implements regularized linear models with stochastic gradient descent (SGD) learning: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate).

This implementation works with data represented as dense or sparse arrays of floating point values for the features. The model it fits can be controlled with the loss parameter; by default, it fits a linear support vector machine (SVM).



```
from sklearn.linear_model import SGDClassifier
ext_clf_SGD = Pipeline([('vect', TfidfVectorizer(ngram_range=(1, 2), stop_words=stop_words_ar, sublinear_tf=True)),
                    ('chi',  SelectKBest(chi2, k='all')),
                    ('clf', SGDClassifier())])
```

6 Results

The final results on the testing dataset are as follow on Arabic Dataset

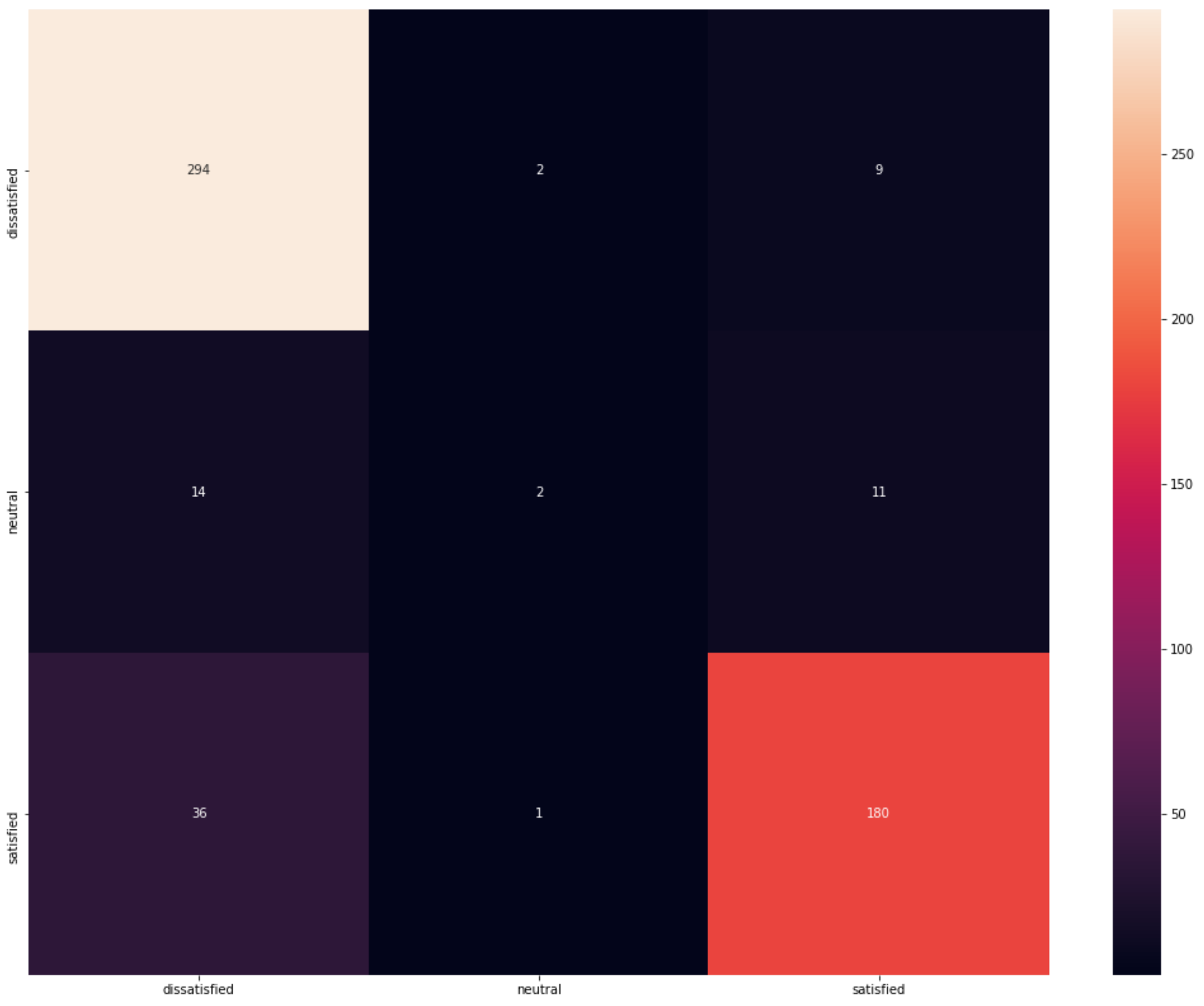
Classifier	Accuracy
SCV	72%
KNN	79%
NB	88%
XGBoost	89%
SGD	90%

SGD on English Dataset reached 99% using SGD

6.1 Confusion Matrix

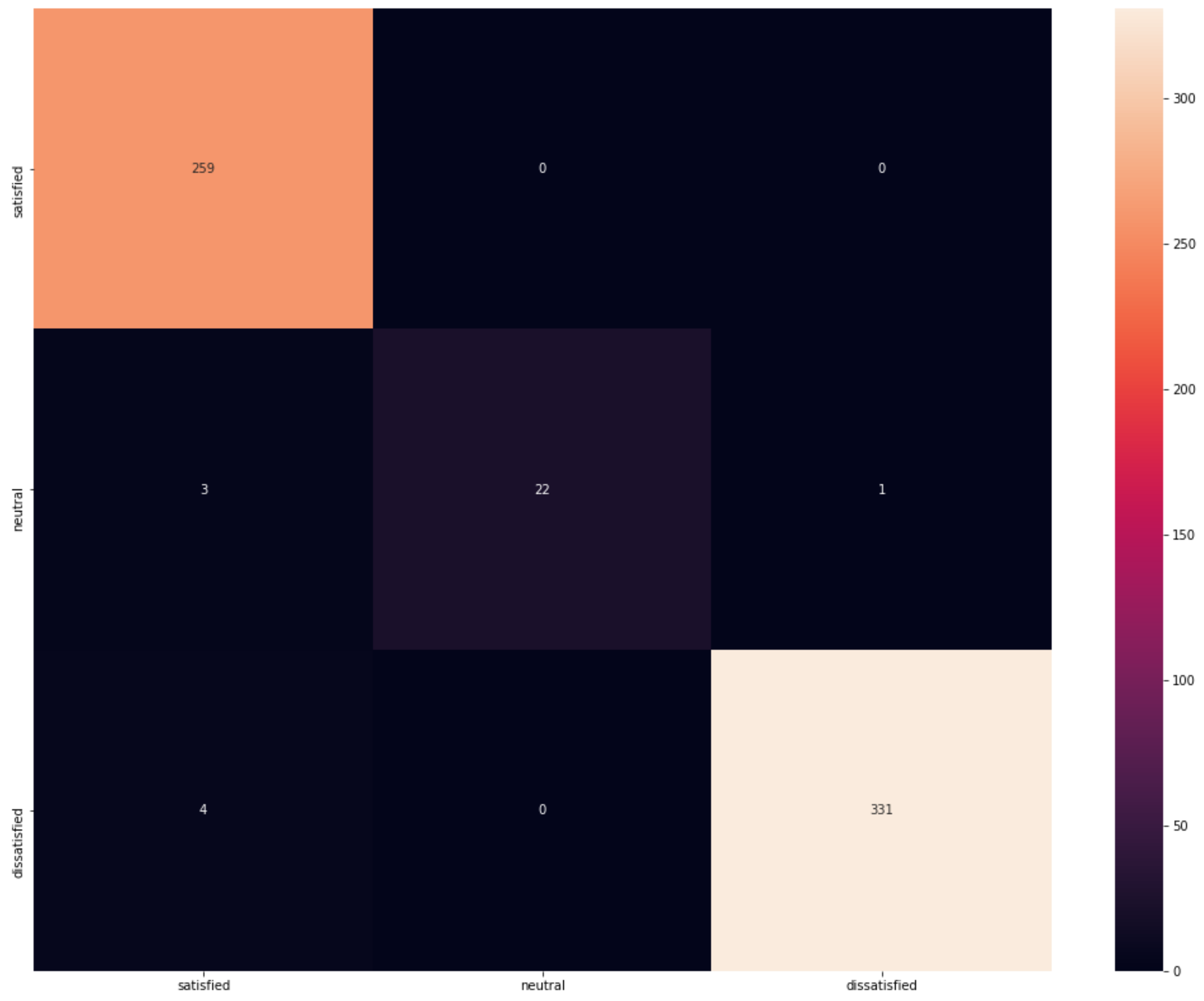
As The Accuracy in classifier models isn't enough estimate of how the model is good or not. I used the f1 (Precision and Recall) and built the confusion matrix[2] as follow.

6.1.1 Arabic Model Testing



[2] is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one.The confusion matrix shows the ways in which your classification model is confused when it makes predictions. It gives us insight not only into the errors being made by a classifier but more importantly the types of errors that are being made.

6.1.2 English Model Testing



As we can see, the models are great on the testing samples. As each class sentences are mostly predicted correct.

And here we go, we now have a model that works fine for our task

Now lets save the model to use it in the future.

```
import pickle
filename = '/content/en_finalized_model.sav'
pickle.dump(SGD_model, open(filename, 'wb'))
```

7 How to use?

Simply Upload the 2 models provided, and the Reviews to the notebook (also provided) and the code will separate Arabic and English reviews and each one will go to the 2 models. The models will generate if the user is satisfied or not, for example

```
print(SGD_model.predict(['i hate it', 'i like the app', 'very bad', 'Awesome']))
```

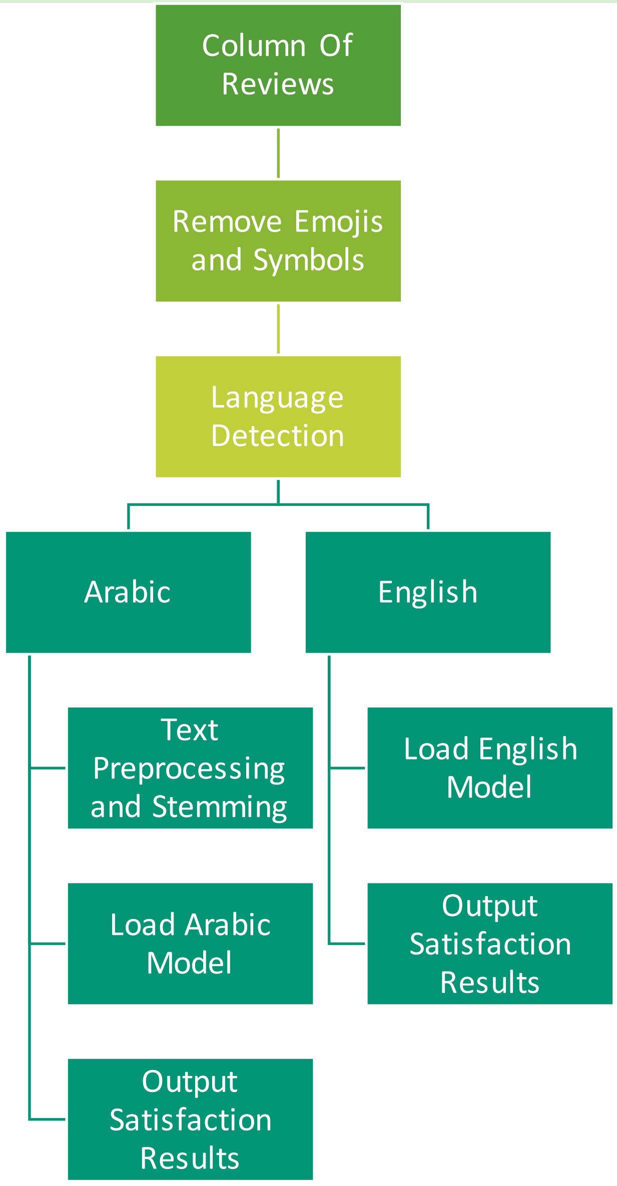
```
['dissatisfied' 'satisfied' 'dissatisfied' 'satisfied']
```

```
print(SGD_model.predict(['وحش', 'ممتاز', 'سيء جدا', 'مش حلو']))
```

```
['dissatisfied' 'satisfied' 'dissatisfied' 'dissatisfied']
```

Here as we can see it gets all the reviews right.

7.1 System Flow



Column Of Reviews
Remove Emojis and Symbols
Language Detection
Arabic
English
Text Preprocessing and Stemming
Load Arabic Model
Output Satisfaction Results
Load English Model
Output Satisfaction Results

Notes:

- The Excel Sheet of reviews should be named (reviews.xlsx). And should be uploaded to The Notebook.
- Upload the 2 Models (Arabic Model – English Model) to The Notebook.
- You Should Upload the Font File for the Arabic WordCloud.
- The Column of reviews should be named (reviews).
- The output will be 4 WordClouds of (Satisfaction and Dissatisfaction in English and Arabic) 6 Excel Sheets:
  - Arabic Results
  - English Results
  - Arabic Satisfied Users
  - Arabic Dissatisfied Users
  - English Satisfied Users
  - English Dissatisfied Users

Notebook Link: Link

NLP Models and Font File Link: Link

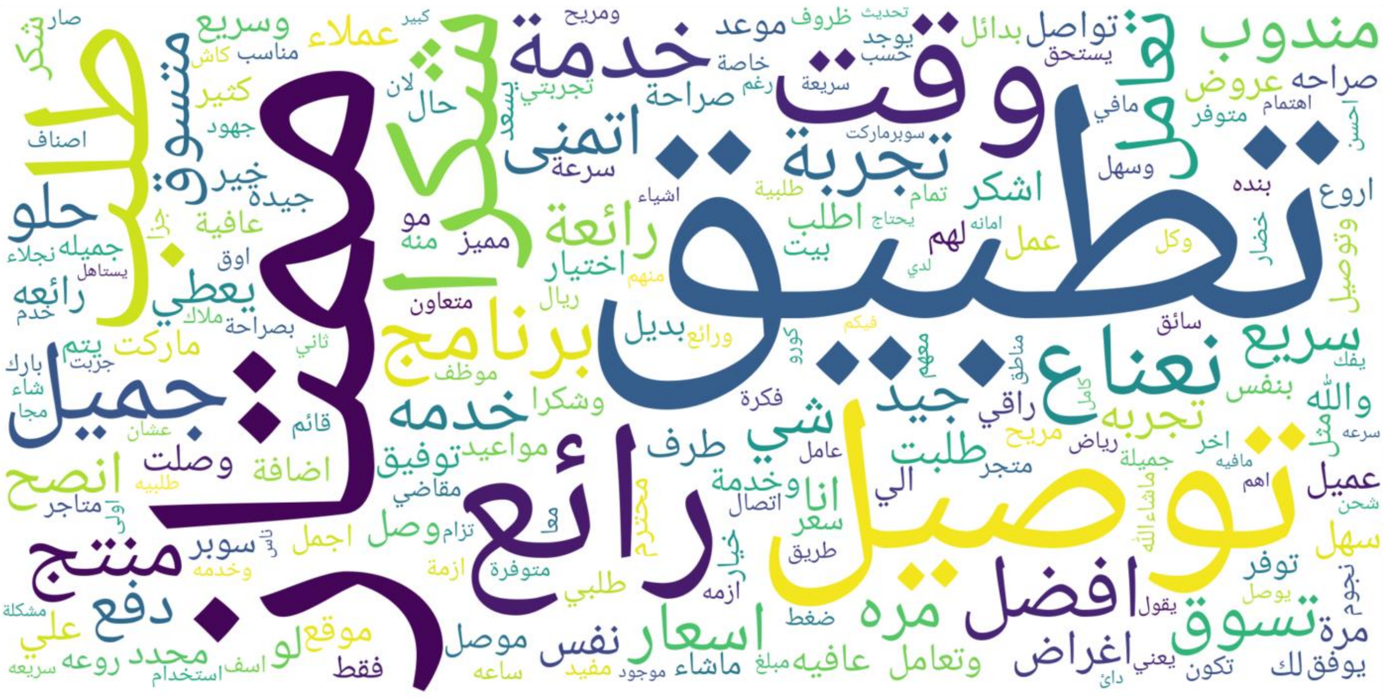
8 Results On the Received DataSet

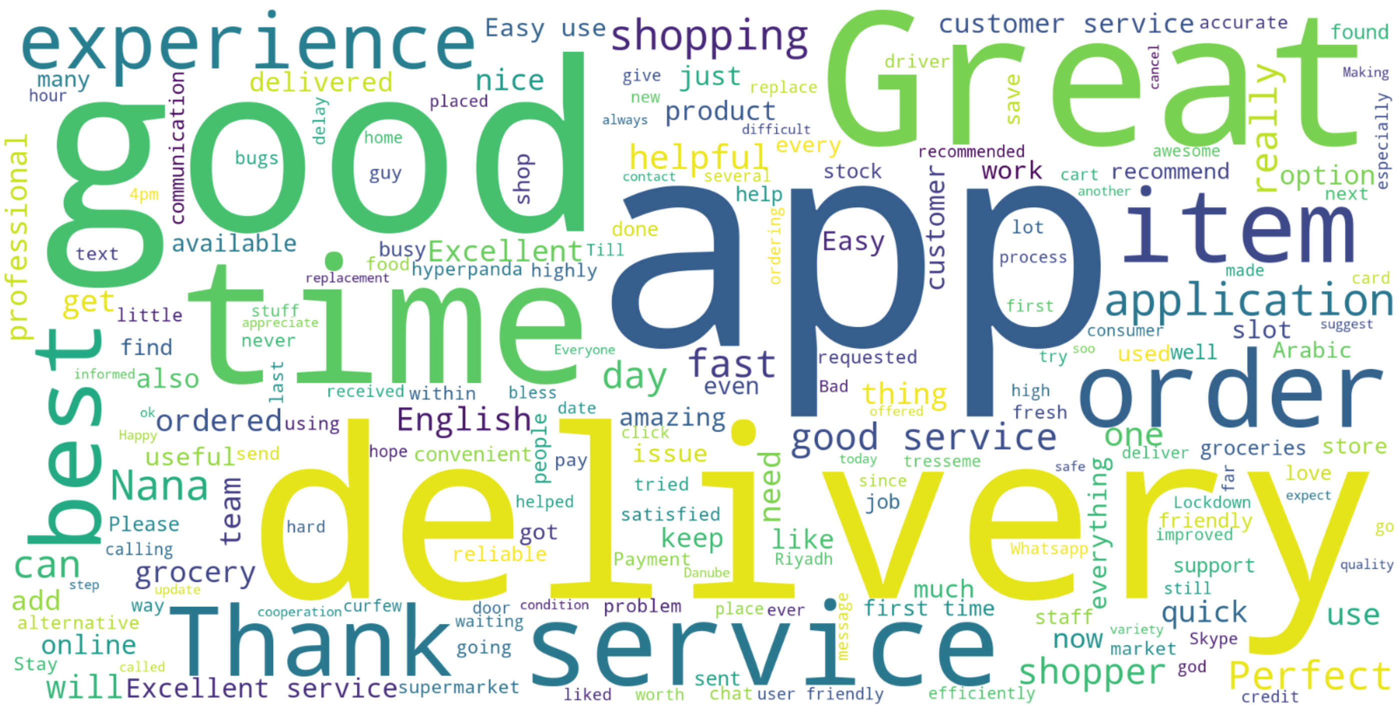
Language / Sentence Count	Satisfied	Dissatisfied
Arabic	2171	3086
English	287	363

8.1 Why The Customer Is Satisfied or not?

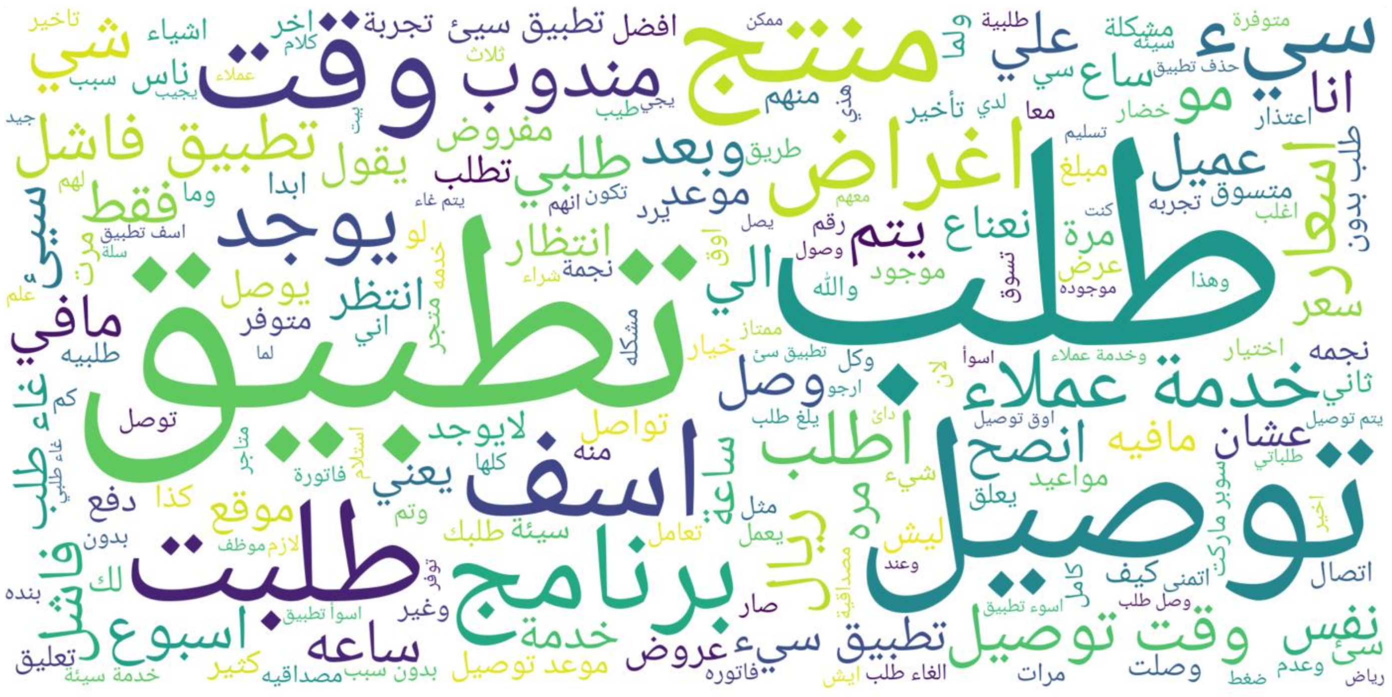
To measure if the customer is satisfied, we will see using WordCloud, the most words and scentences that indicates his satisfication.

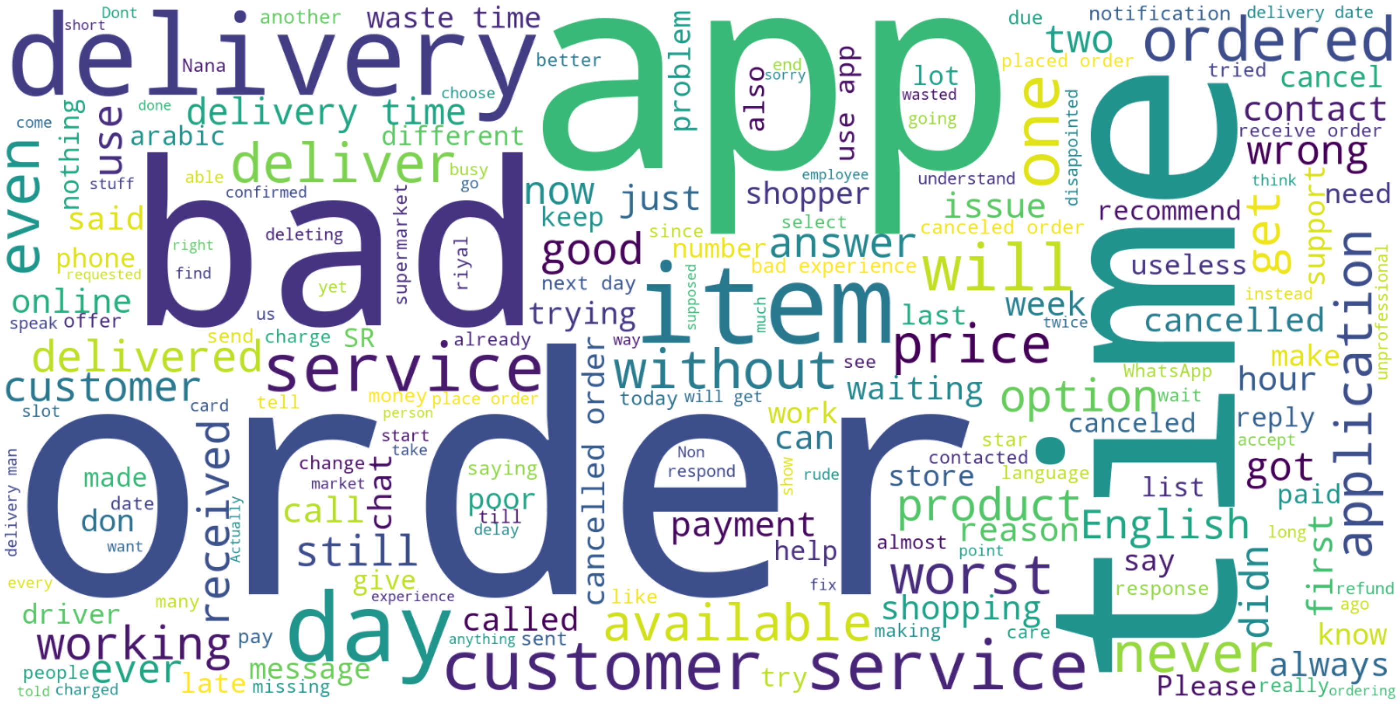
8.1.1 Satisfied WordClouds





8.1.2 Dissatisfied WordClouds





Now as we see (Time – Price – Ordering – and Customer Service – Cancelled Orders – Limited Products) are the main reasons for dissatisfaction of the users.

9 References

- Detecting bad customer reviews with NLP | by Jonathan Oheix | Towards Data Science
- Classifying Movie Reviews With Natural Language Framework | by Anupam Chugh | Towards Data Science
- Reviews Classification NLP | Kaggle
- An NLP Tutorial for Text Classification | Toptal
- Text Analysis and Classification of Reviews in Python | by Abir | Medium
- saobou/arabic-text-preprocessing: In this notebook you will find all the functions that we use to process a text
- motazsaad/process-arabic-text: Pre-process arabic text (remove diacritics, punctuations and repeating characters)
- motazsaad/arabic-light-stemming-py: Arabic Light stemming with Python
- bakrianoo/aravec at 11457255d7a6114da7ff5e16f3c936eeccaa672a
- A Beginner's guide to XGBoost. This article will have trees.... lots of… | by George Seif | Towards Data Science
- XGBoost Parameters — xgboost 1.2.0-SNAPSHOT documentation
- Grid Search for Hyperparameter Tuning | by Mathanraj Sharma | Towards Data Science
- XGBoost hyperparameter tuning in Python using grid search | Bartosz Mikulski
- FantacherJOY/Arabic-text-classification: Arabic text documents classified using SVM, k-nn and Naive bayes classifers.
- Multi-Class Text Classification with Scikit-Learn | by Susan Li | Towards Data Science
- Text Classification with Python and Scikit-Learn
- TF IDF | TFIDF Python Example. An example of how to implement TFIDF… | by Cory Maklin | Towards Data Science
- Sentiment Analysis in Arabic tweets using sklearn | Kaggle
- Multi-Class Text Classification Model Comparison and Selection
- sklearn.naive_bayes.MultinomialNB — scikit-learn 0.23.2 documentation
- jasonwei20/eda_nlp: Data augmentation for NLP, presented at EMNLP 2019
- Machine Learning, NLP: Text Classification using scikit-learn, python and NLTK. | by Javed Shaikh | Towards Data Science
- Save and Load Machine Learning Models in Python with scikit-learn
- KNN Classification
- (Tutorial) Support Vector Machines (SVM) in Scikit-learn - DataCamp
- Naive Bayes classifier - Wikipedia
- sklearn.linear_model.SGDClassifier — scikit-learn 0.23.2 documentation
- Confusion Matrix in Machine Learning - GeeksforGeeks