Grammar and Product reviews

Data Mining

Prof. Dr. Ing: Lule Ahmedi
Ass: Vigan Abdurrahmani

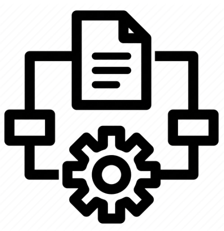
Students:
Besian Shabani
Dafina Imeraj
Fortesa Hysenaj

Overview

Text Preprocessing

Statistics

WordCloud



Algorithms we used

Prototype Description

Introduction

Natural Language Processing (NLP) is ubiquitous and has multiple applications. For example, a few use cases include email classification into spam and ham, chatbots, AI agents, social media analysis, and classifying customer or employee feedback as positive, negative, or neutral.

Despite these many applications of NLP, implementation remains difficult because text data is different from tabular transactional data. Cleaning text data is also difficult because you deal with natural language. This is where text processing is helpful to clean the text corpus and make it ready for further operation.

Preprocessing

It is important to pre-process text before you run the module to extract key phrases from the corpus. The most common pre-processing steps are:

1. Remove stop words: These are unhelpful words like 'the', 'is', 'at'. They are not helpful because the frequency of such stop words is high in the corpus, but they don't help in differentiating the target classes. The removal of stop words also reduces the data size.
2. Detect Sentences: This inserts a sentence boundary mark while performing analysis. The sentence terminator is represented by three pipe characters: |||.
3. Remove punctuation: The rule of thumb is to remove everything that is not in the form of x,y,z.
4. Normalize case to lowercase: Words like 'Clinical' and 'clinical' need to be considered as one word, so they are converted to lowercase.
5. Remove special characters: Non-alphanumeric special characters are replaced with the pipe | character.
6. Expand verb contractions: This is an important feature applied to verb contractions. For example, selecting this option will replace the phrase "wouldn't buy this product" with "would not buy this product".
7. Stemming: The goal of stemming is to reduce the number of inflectional forms of words appearing in the text. This causes words such as “argue,” "argued," "arguing," and "argues" to be reduced to their common stem, “argu”. This helps decrease the size of the vocabulary space.

Preprocessing

```
stop_words = set(line.strip() for line in open('stopwords.txt'))
exclude = set(string.punctuation)
def docs_preprocessor(docs):
    tokenizer = RegexpTokenizer(r'\w+')
    for idx in range(len(docs)):
        docs[idx] = docs[idx].lower()  # Convert to lowercase.
        docs[idx] = tokenizer.tokenize(docs[idx])  # Split into words.
    # Remove numbers, but not words that contain numbers.
    docs = [[token for token in doc if not token.isdigit()] for doc in docs]
    #Remove stop words in documents
    docs = [[token for token in doc if not token in stop_words] for doc in docs]
    #Remove punctuations in documents
    docs = [[token for token in doc if not token in exclude] for doc in docs]
    # Remove words that are only one character.
    docs = [[token for token in doc if len(token) > 2] for doc in docs]

    # Lemmatize all words in documents.
    lemmatizer = WordNetLemmatizer()
    docs = [[lemmatizer.lemmatize(token) for token in doc] for doc in docs]

    return docs
# Perform function on our document
documents = docs_preprocessor(docs)
```

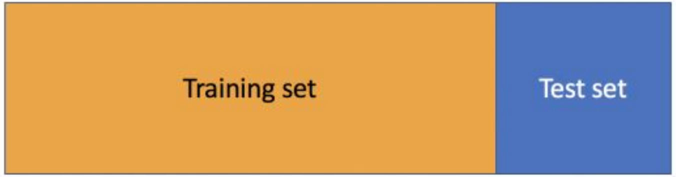
Text cleaning is one of the most challenging areas of natural language processing. In every application of text analytics, such as email classification, sentiment analysis, key phrase extraction, or visualizing text data, you will be required to perform text cleaning.

Statistics

- 25 columns
- 71045 tuples

Most important columns:
- Brand
- Categories
- Reviews-purchase
- Reviews-recommended

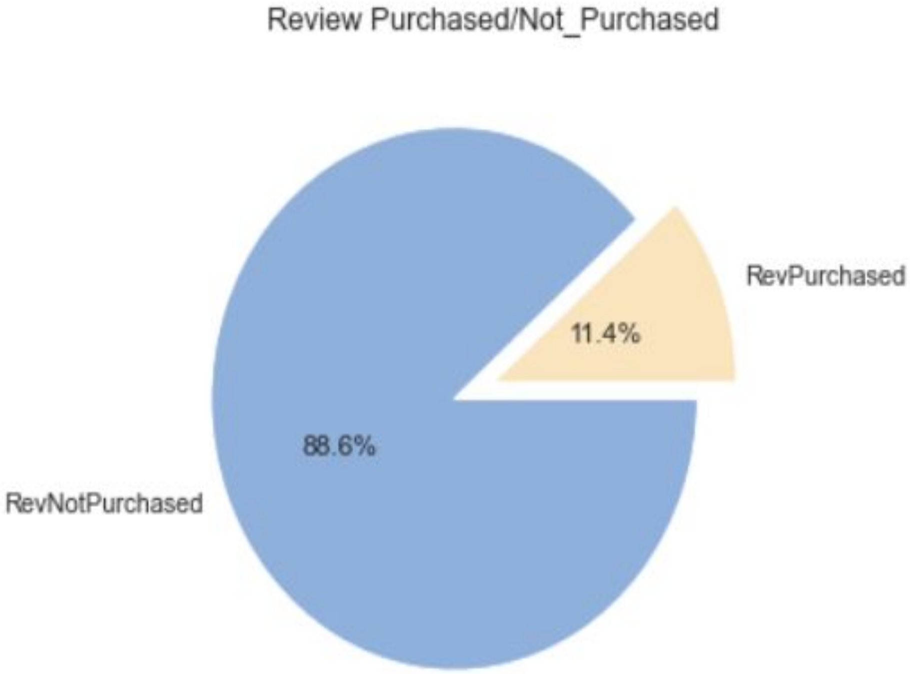
Training set 80%
Testing set 20%



https://www.kaggle.com/datafiniti/grammar-and-online-product-reviews

Review Purchased and Not Purchased

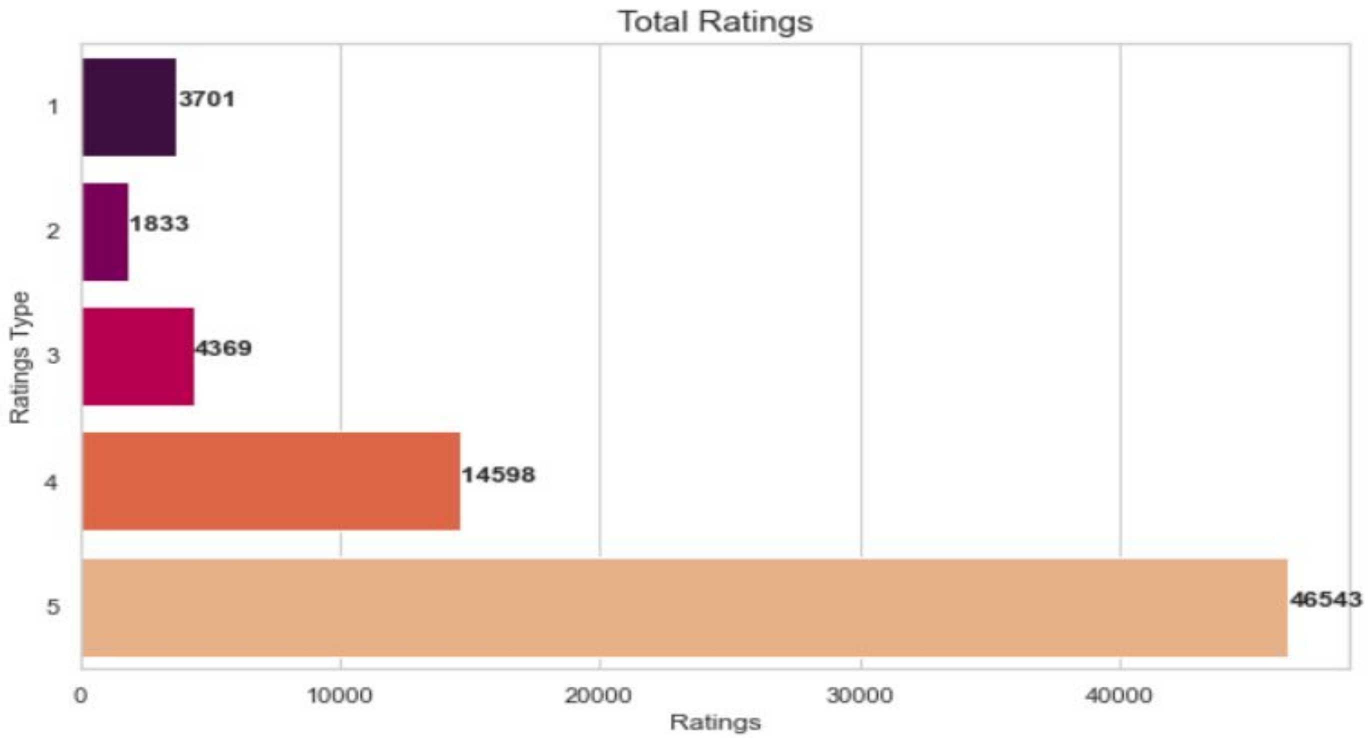
	Total
RevPurchasedTrue	3682
RevPurchasedFalse	28476



Review Recommended and Not Recommended

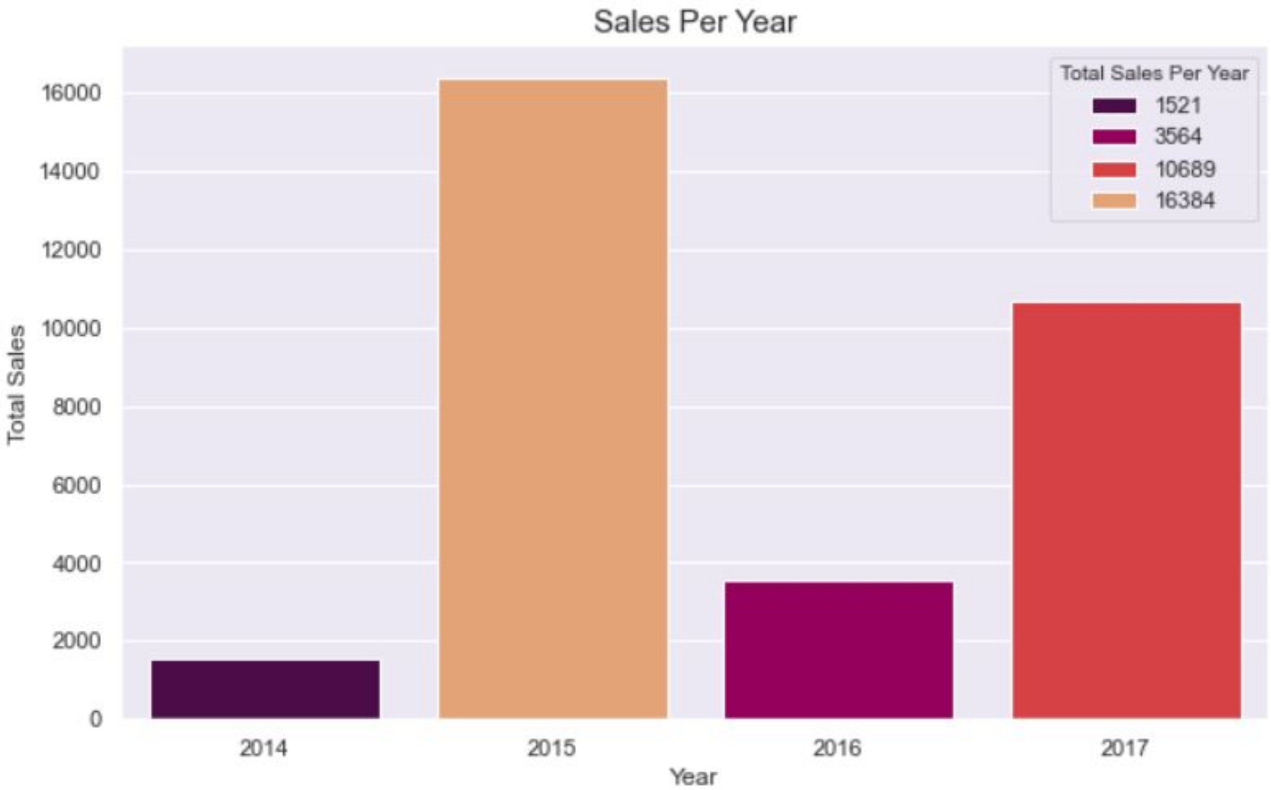
	Total
RevRecommended	55587
RevNotRecommended	4842

Reviews Rating



Total Sales per Year

Year	Total Sales Per Year
2014	1521
2015	16384
2016	3564
2017	10689



WordCloud

Text data has grown exponentially in recent years resulting in an ever-increasing need to analyze the massive amounts of such data. Word Cloud provides an excellent option to analyze the text data through visualization in the form of tags, or words, where the importance of a word is explained by its frequency.

WordCloud

The first line of code generates the word cloud on the 'reviewText' corpus, while the second to fourth lines of code prints the word cloud.

```
wordcloud = WordCloud(width=800, height=500, max_words=100, background_color="black", min_word_length=5).generate(reviewText)

plt.axis("off")
plt.imshow(wordcloud, interpolation='bilinear')
plt.show()

wordcloud.to_file("wordcloud.jpeg")
```

Output:

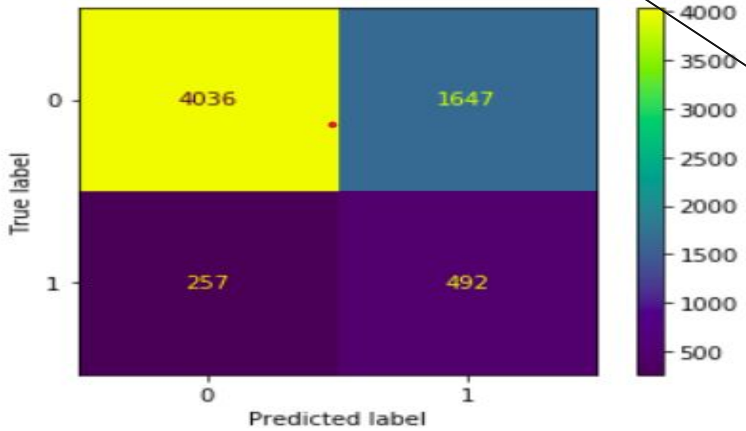
The word cloud displayed here is good, but some of the words are larger than the others. This is because the size of the word in the word cloud is proportional to the frequency of the word inside the corpus. There are various parameters which can be adjusted to change the display of the word cloud, and the list of such parameters can be viewed using the '?WordCloud' command.

Naive Bayes Algorithm

Results from Naive Bayes

	precision	recall	f1-score	support
0	0.9401	0.7102	0.8091	5683
1	0.2300	0.6569	0.3407	749
accuracy			0.7040	6432
macro avg	0.5851	0.6835	0.5749	6432
weighted avg	0.8574	0.7040	0.7546	6432

NB accuracy: 0.7039800995024875



We use Byes' theorem to find which label is most likely, given the attributes observed in the feature vector, and given how often the different labels occur in the data.

The Naive Bayes classifier can be an extremely powerful one, and it often results in very robust models.

It's called Naive because the different X variables, that is, the different feature in our dataset are consider to be independent.

This model performed decently well, and its accuracy on the test data was around 70%.

Decision tree algorithm

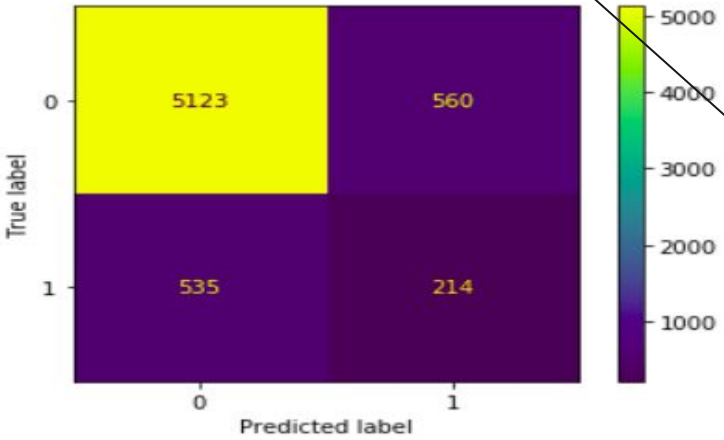
```
Results from Decision Tree

              precision    recall  f1-score   support

           0     0.9054    0.9015    0.9034      5683
           1     0.2765    0.2857    0.2810       749

    accuracy                         0.8298      6432
   macro avg     0.5910    0.5936    0.5922      6432
weighted avg     0.8322    0.8298    0.8310      6432

DT accuracy: 0.8297574626865671
```



The decision tree is a popular and widely used machine learning model for classification problems.

You can build and train a decision tree for classification models in scikit-learn using the decision tree classifier.

We have fit a decision tree on our training data using the CART(Classification and Regression Tree) algorithm.

A decision tree splits your underlying data into subsets where every subset contains points that is consider similar. The data is repeatedly split into subsets to form a tree structure, and the shape of the tree depends on the constraints that you specify.

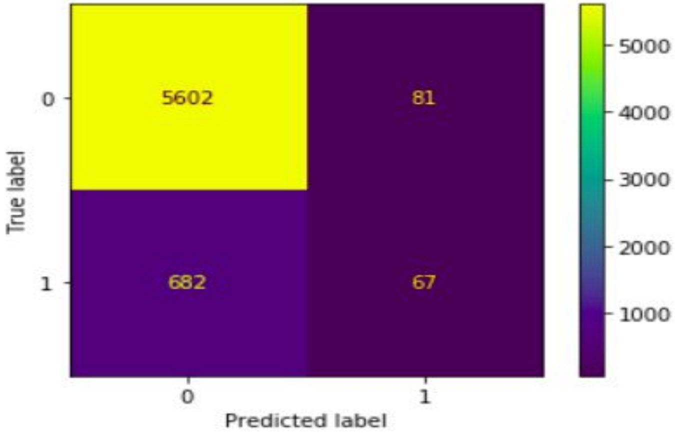
Here you can see that even though on test data, the model accuracy is 82%, which is around what we got with other models, on training data it's clearly overfitted, so it's not really a very good model.

Boosting Algorithm

Results from Boosting

	precision	recall	f1-score	support
0	0.8915	0.9857	0.9362	5683
1	0.4527	0.0895	0.1494	749
accuracy			0.8814	6432
macro avg	0.6721	0.5376	0.5428	6432
weighted avg	0.8404	0.8814	0.8446	6432

AdaBoost accuracy: 0.8813743781094527



In Boosting, multiple models are trained sequentially and each model learns from the errors of its predecessors. In this guide, we will implement one boosting techniques of AdaBoost.

AdaBoost, short for 'Adaptive Boosting', is the first practical boosting algorithm proposed by Freund and Schapire in 1996. It focuses on classification problems and aims to convert a set of weak classifiers into a strong one.

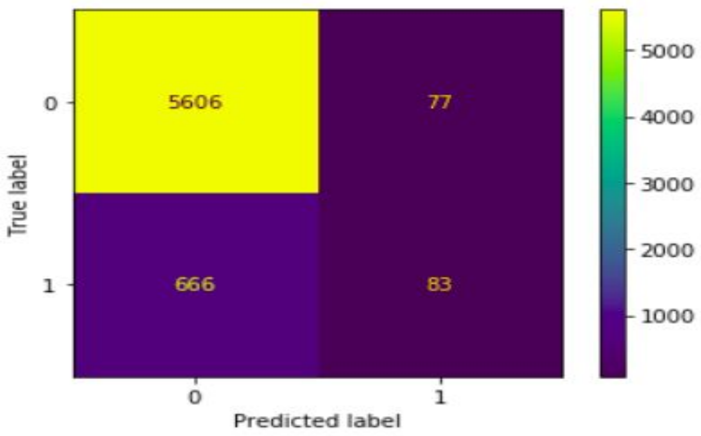
The accuracy of the AdaBoost Classifier ensemble is 88%, which is higher than the other models.

Random Forest Algorithm

Results from Random Forest

	precision	recall	f1-score	support
0	0.8938	0.9865	0.9379	5683
1	0.5188	0.1108	0.1826	749
accuracy			0.8845	6432
macro avg	0.7063	0.5486	0.5602	6432
weighted avg	0.8501	0.8845	0.8499	6432

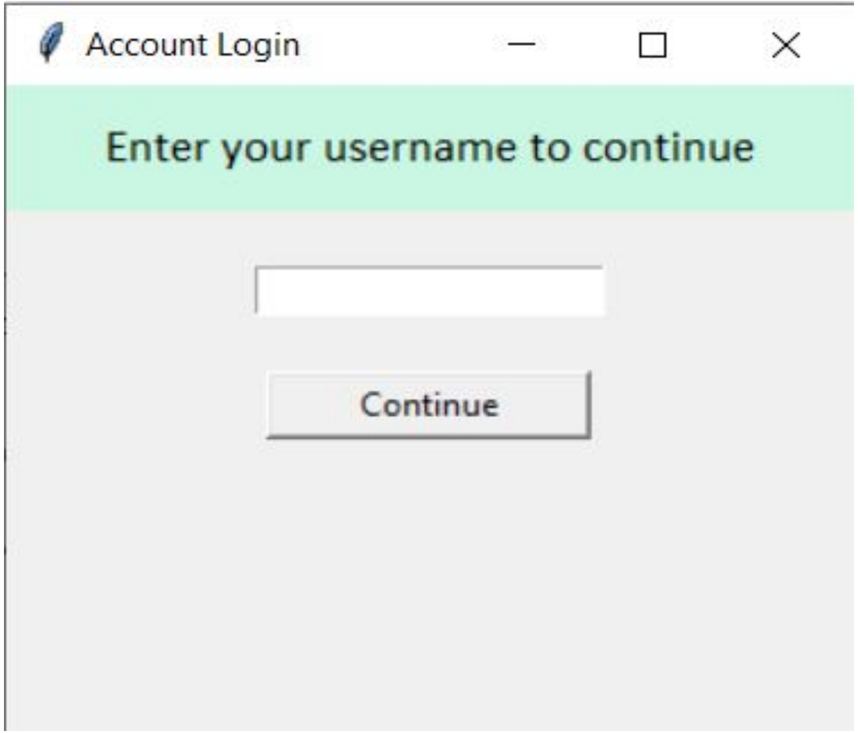
$\boxed{\text{Random Forest accuracy: 0.8844838308457711}}$



Random Forest is an extension of bagged decision trees, where the samples of the training dataset are taken with replacement. The trees are constructed with the objective of reducing the correlation between the individual decision trees. In scikit-learn, a random forest model is constructed by using the Random Forest Classifier class.

The accuracy of the Random Forest Classifier ensemble is 88.44%, a significant improvement over the other models.

Prototype Description

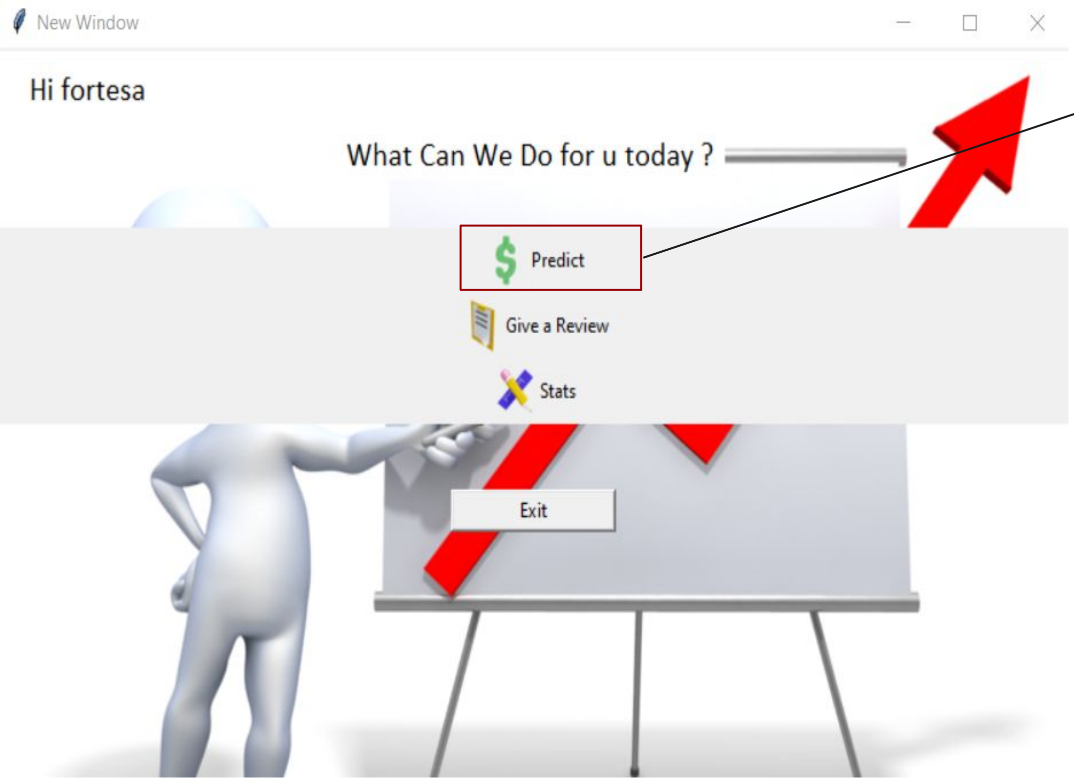


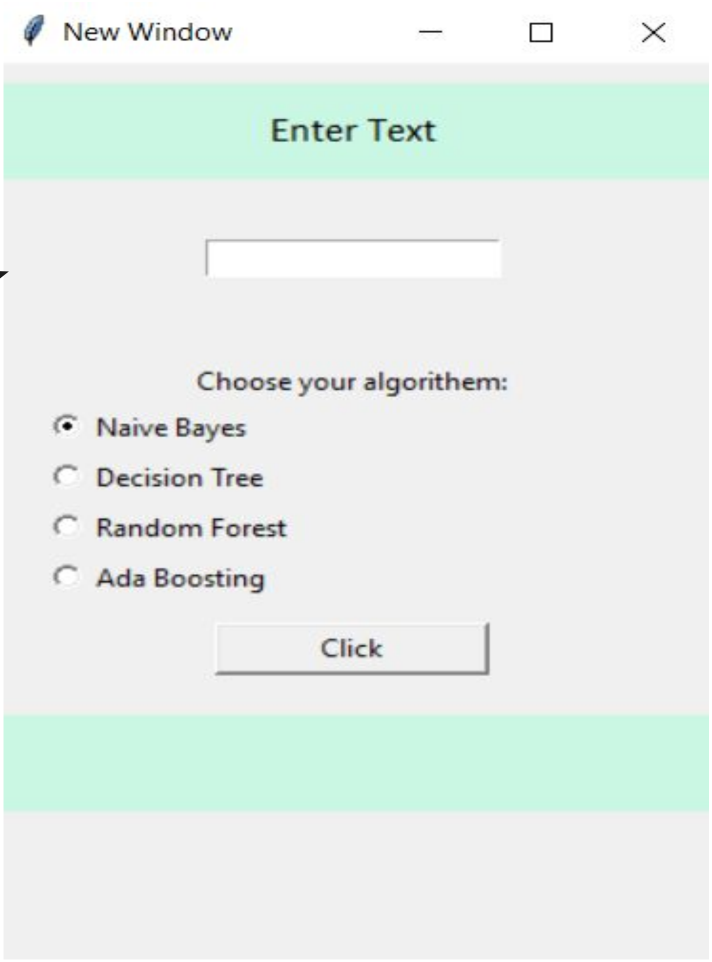
Our GUI Form

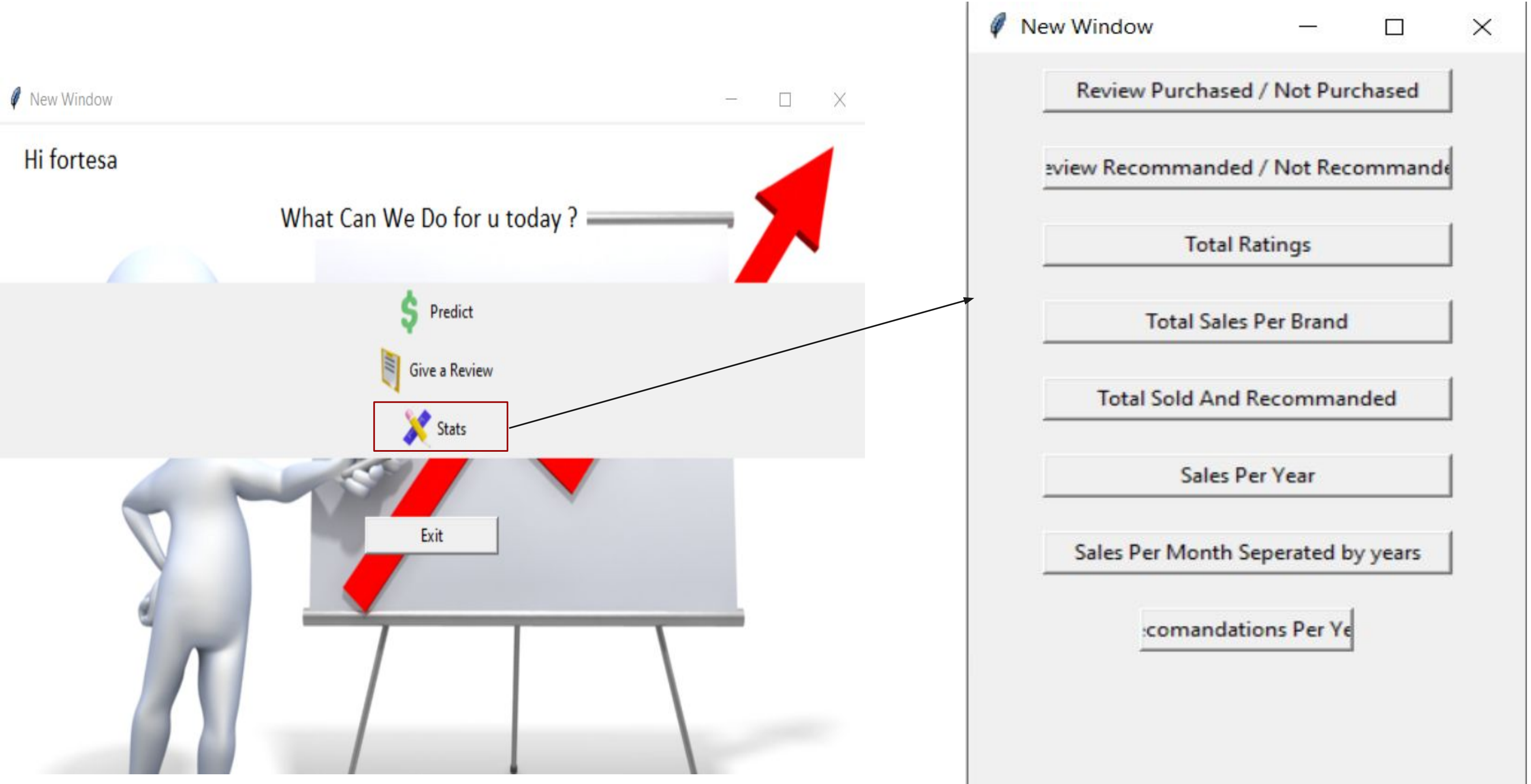
Prediction

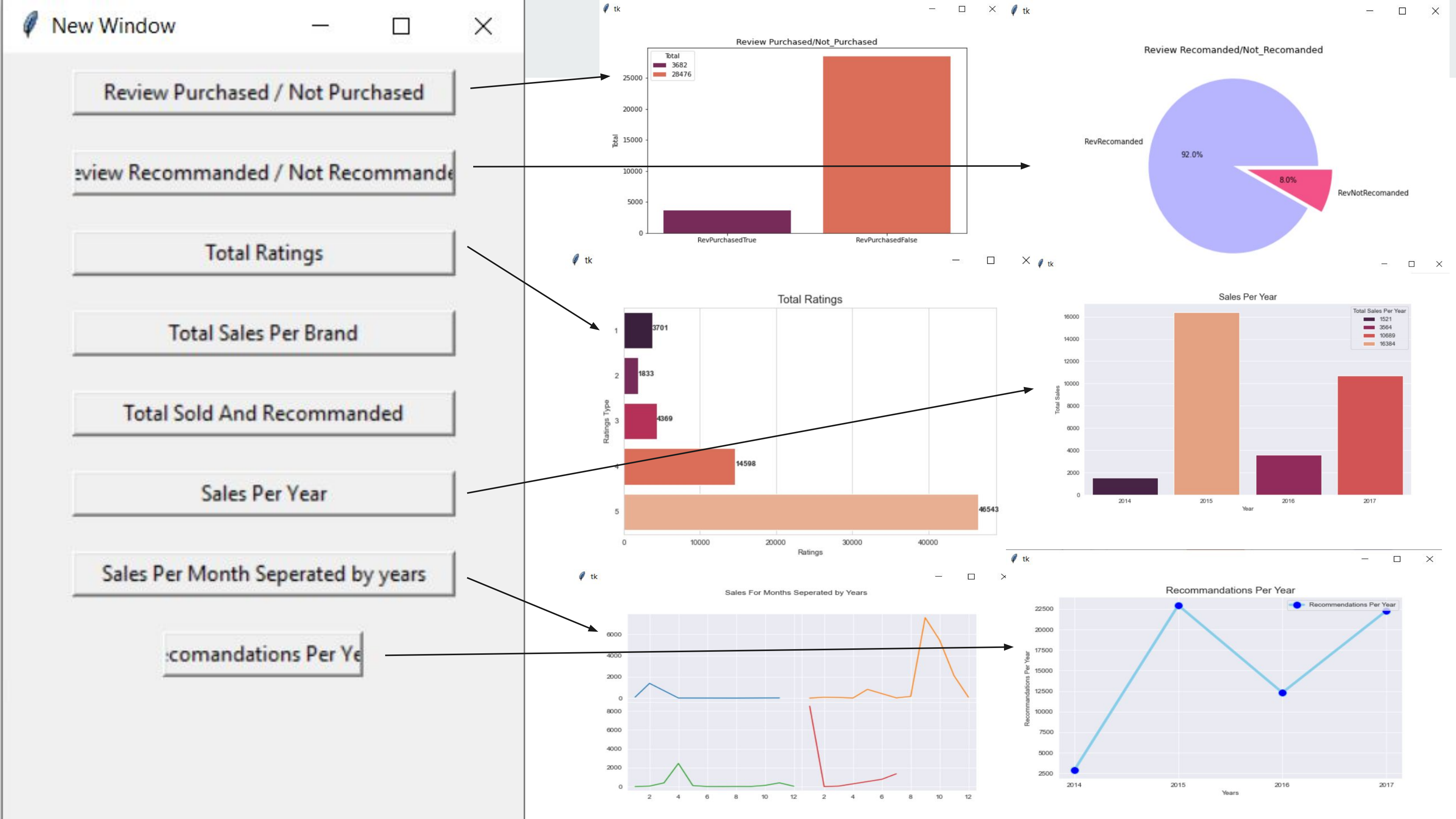
Statistics

GUI Form


New Window
Hi fortesa
What Can We Do for u today ?
Predict
Give a Review
Stats
Exit


New Window
Enter Text
Choose your algorithem:
Naive Bayes
Decision Tree
Random Forest
Ada Boosting
Click


New Window
Hi fortesa
What Can We Do for u today ?
Predict
Give a Review
Stats
Exit
New Window
Review Purchased / Not Purchased
eview Recommanded / Not Recommande
Total Ratings
Total Sales Per Brand
Total Sold And Recommanded
Sales Per Year
Sales Per Month Seperated by years
comandations Per Ye



New Window
Review Purchased / Not Purchased
eview Recommanded / Not Recommande
Total Ratings
Total Sales Per Brand
Total Sold And Recommanded
Sales Per Year
Sales Per Month Seperated by years
comandations Per Ye
tk
Review Purchased/Not_Purchased
Total
3682
28476
RevPurchasedTrue
RevPurchasedFalse
Review Recomanded/Not_Recomanded
RevRecomanded
92.0%
8.0%
RevNotRecomanded
Total Ratings
Ratings Type
3701
1833
4369
14598
46543
Ratings
Sales Per Year
Total Sales Per Year
1521
3564
10689
16384
Total Sales
Year
Sales For Months Seperated by Years
Recommandations Per Year
Recommendations Per Year
Recommandations Per Year
Years