Reinvent Retail - QFree
Machine Intelligence with Python

Manojit Chakraborty*

Academic Intern
Tata Consultancy Services
Gitanjali Park, Kolkata

B.Tech, Computer Science and Engineering
Heritage Institute of Technology, Kolkata

Dated: August 8, 2017

*Electronic address: `manojit.chakraborty1@tcs.com`

Contents

1 Introduction

1.1 Problem Statement

Create a Smart Retail Store App using modern artificial intelligence techniques, that can give the customer(s) a hassle-free shopping experience and identify the fastest moving queue for checkout inside a retail store in real time.

1.2 Background

Retail stores deal with a large number of customers every day. The checkout lines are considered to be one of the most critical junctures at a retail outlet. However, retailers are finding it hard to manage these checkout queues effectively. Also, a lot of retailers believe that once a customer enters into checkout line, the sale is almost won. But, that usually isn't the case as customers can still get frustrated and annoyed while waiting in long checkout lines.

Research suggests that customers won't consider returning to a retail outlet that has unmanaged queues and creates unpleasant experience. In order to create pleasant customer experiences and gain customer loyalty, retail outlets must integrate their existing customer checkout lines with a queue management system that ensures that customers are served in a fair manner and waiting lines move smoothly. 80% customers switch to a different retailer for shorter queues. Hence, retailers must try to understand the importance of a having an efficient queue management solution for customers.

The checkout experience creates a lasting impression and effective queue management helps promote customer loyalty and a positive image for the business. Wait times can be prominently displayed for customers to see, inside and outside the store. This is really useful at busy times like lunch breaks when customers are far more likely to enter an apparently busy store if they know in advance, that they will not have to queue for more than a couple of minutes.

1.3 Challenge

- Create a recommendation system which can show the customers -
  - Real Time Trending Products
  - User-based recommendations
  - Item-based recommendations
- Create an intercative chatbot which can converse with the customer and help in every way while shopping.
- Create an algorithm which will identify the fastest moving queue for checkout inside a retail store in real time.
- Create a Sentiment Analyzer which will analyze the shopping feedbacks from the customers.

2 Our Solution

2.1 High-Level Approach

- An app that has virtual cart.
- When the customer enters a store, the app should automatically send notification to let it open OR user may manually open the app.
- The app then initialize a virtual cart for the shopping session.
- Customer takes a product and scans it on the app (e.g. using bar code/qr code) and the item will be added on his virtual cart on the app.
- This way the app keeps track of all products chosen by the customer.
- After shopping is over, the app should assign the customer a checkout counter number to go, based on current traffic on checkout lines (using machine learning and AI).
- Whenever a customer is suggested with a pre- assigned counter number, then his checkout will only be possible from that counter and no other else. (This controls the entire checkout queue)
- If at runtime, one counter becomes faulty, then the app should assign the pending customers to other open counters based on current traffic and priority.

- When its turn for that customer, he should show the barcode/qr-code (generated by the app for that virtual cart) in front of the cashier, and the cashier should then scan the code from the customer and the cart item details will be passed on to the cashier's screen and he proceeds with billing, payout and packing.

- Once the customer leaves the counter, his virtual cart session will be cleared out, and the current successful shopping details will be stored in his account.

- Customer sentimental analysis based on shopping experience and try to make the satisfaction level to the maximum possible.

2.2 Additional Features

- The app will have a section like an interactive bot. it will help customer locating any product in the store, helping throughout the shopping in any way possible.

- It should understand the shopping behaviour of the customer, and suggest him best/cheap/popular products based on items he has bought both in past and current cart. Recommendation system is on item based as well as user based.

- The app's backend Machine learning APIs, REST APIs and other functionalities will be hosted from cloud (e.g. Google cloud / AWS cloud/ MS Azure etc.)

2.3 Architecture

- Python programming for App back-end development.
- Heroku / MS Azure / Google Cloud Platform / AWS for hosting app backend.
- Ionic framework for App front-end development

2.4 Target Retailers

Big Retail Chains like Walmart, Target, Big Bazar, Spencers.

2.5 Assumptions

- Customers will use the app on their internet-enabled smartphones, everytime they enter into the store.
- Well maintained security system that is enabled to detect any fraud incase customer tries to go outof the store without checking out a product.

2.6 Expected Benefits

- Checkout queue will be faster and under control at everytime.
- Customer shopping experience will raise highly.
- Repeat business probability from same customer will increase significantly.
- Issue solutions at checkout lines will be better.

3 My Project Work

3.1 Challenges

- Create and populate the following databases for running the prototype app - Product, User, Temporary Cart, Purchase cart, Reviews, Offers, Location, Wish list.
- Create the Algorithm for assigning the Fastest Checkout Queue in Real-Time.
- Create a Recommendation System module using collaborative filtering, which will have 3 sub-modules
  - Trending Item Finder
  - User-based recommendation engine
  - Item-based recommendation engine
- Create a Sentiment Analyzer module that will analyze customer sentiments through feedback and chat messages, giving state-of-the-art accuracy on the algorithm.
- Create an Interactive Chatbot module which will help a customer in any way possible while shopping, such as finding a product, giving recommendations, showing trending items, start or finish the shopping, helping to assign the fastest checkout queue.
- Create Item Finder module which will return the full location of any item inside the store, asked by customers.
- Create a Computer Vision module which will automatically detect the faces of customers standing in front of the cashiers.
- Create an Integration and Testing module which will take input a JSON file from Node.JS and return outputs accordingly.

3.2 Implementations

3.2.1 Database

Database is one of the most important parts for implementing machine intelligence on a product. That's why first importance was given to create the databases and populate them with prototype data. Then the .db files are converted to .csv file for ease of use in further processes.

Language used Python

Packages used sqlite3, pandas

Code Here is the code for creating the Products database. All the other databases have been created using similar code.

```
# -*- coding: utf-8 -*-
"""
Created on Thu Jun 22 11:58:15 2017

@author: 1399869
"""

import sqlite3
import pandas as pd

conn = sqlite3.connect('Products.db')
print "Opened database successfully";

conn.execute('''CREATE TABLE Products (
  product_id integer PRIMARY KEY AUTOINCREMENT,
  product_name text,
  price decimal,
  weight decimal,
  pack_time decimal,
  unit_type text,
  picture_url text,
  product_code integer,
  product_count integer,
  floor_no integer,
  section text,
  rack_no integer
  );''')
```

```python
print "Table created successfully";



df = pd.read_sql("SELECT * from Products",con=conn)
df.to_csv("Products.csv")

conn.close()
```

Listing 1: database.py

Let's look at the table now for information.

```
In [1]: import pandas as pd

In [2]: df = pd.read_csv("Products.csv")

In [4]: df.head(10)
```

Out[4]:

product_id	product_name	price	weight	pack_time	unit_type	picture_url	product_code	product_count	floor_no	section	rack_
1	abrasive-cleaner	1	1	1	1	1	12345600	10	1	home_appliances	1
2	artif.-sweetener	2	2	2	2	2	12345601	10	1	groceries	2
3	baby-cosmetics	3	3	3	3	3	12345602	10	1	cosmetics_personal_care	3
4	baby-food	4	4	4	4	4	12345603	10	1	groceries	4
5	bags	5	5	5	5	5	12345604	10	1	groceries	5
6	baking-powder	6	6	6	6	6	12345605	10	1	groceries	6
7	bathroom-cleaner	7	7	7	7	7	12345606	10	1	home_appliances	7
8	beef	8	8	8	8	8	12345607	10	1	groceries	8
9	berries	9	9	9	9	9	12345608	10	1	groceries	9
10	beverages	10	10	10	10	10	12345609	10	1	energy_drink	10

The other databases that has been created are -

- User
- Temporary Cart
- Purchase cart
- Reviews
- Offers
- Location
- Wish list

3.2.2 Recommendation Engine

Recommendation System A recommendation system, also known as a recommender system, is software that analyzes available data to make suggestions for something that a user/customer might be interested in, such as a book, a product or a job, among other possibilities.

Collaborative Filtering Collaborative filtering (CF) is a technique used by recommender systems. Applications of collaborative filtering typically involve very large data sets. Collaborative filtering methods have been applied to many different kinds of data including: sensing and monitoring data, such as in mineral exploration, environmental sensing over large areas or multiple sensors; financial data, such as financial service institutions that integrate many financial sources; or in electronic commerce and web applications where the focus is on user data, etc.

Types Collaborative filtering systems have many forms, but many common systems can be reduced to two steps:

- Look for users who share the same rating patterns with the active user (the user whom the prediction is for).
- Use the ratings from those like-minded users found in step 1 to calculate a prediction for the active user

This falls under the category of **user-based collaborative filtering**. A specific application of this is the user-based Nearest Neighbor algorithm.

Alternatively, **item-based collaborative filtering** (users who bought x also bought y), proceeds in an item-centric manner:

- Build an item-item matrix determining relationships between pairs of items
- Infer the tastes of the current user by examining the matrix and matching that user's data

Implementation Our Algorithm implements both user-based collaborative filtering and item-based collaborative filtering to use in different scenarios. It also has a function which detects the most trending items in real-time and show them to users according to their needs.

Language used Python

Packages used pandas, numpy, scipy

Code Here is my code for implementing Recommendation System using user-based collaborative filtering, item-based collaborative filtering and trending items.

```python
# -*- coding: utf-8 -*-
"""
Created on Tue Jun 20 11:37:01 2017

@author: Manojit Chakraborty
"""

#Import libraries
import pandas as pd
from scipy.spatial.distance import cosine
data = pd.read_csv("groceries_2.csv")

#Helper function to get similarity scores
def getScore(history, similarities):
   return sum(history*similarities)/sum(similarities)



def item_based(item):

    data["Quantity"] = 1

    itemarray = pd.unique(data.item)
    #This particular view isn't very helpful for us for analysis.
    #This way of data being arranged is called LONG
    #We need it in wide format
    #Converting data from long to wide format
    dataWide = data.pivot("Person", "item", "Quantity")
    #Replace NA with 0
    dataWide.fillna(0, inplace=True)
```

```
    #Drop the Person column
    data_ib = dataWide.copy()
    data_ib = data_ib.reset_index()
    #Drop the Person column
    #data_ib = data_ib.iloc[:,1:]
    data_ib = data_ib.drop("Person", axis=1)
    # Create a placeholder dataframe listing item vs. item
    data_ibs = pd.DataFrame(index=data_ib.columns, columns=data_ib.columns)
    for i in range(0,len(data_ibs.columns)) :
        # Loop through the columns for each column
        for j in range(0,len(data_ibs.columns)) :
            # Fill in placeholder with cosine similarities
            data_ibs.ix[i,j] = 1-cosine(data_ib.ix[:,i],data_ib.ix[:,j])

    data_neighbours = pd.DataFrame(index=data_ibs.columns,columns=range(1,11))

    # Loop through our similarity dataframe and fill in neighbouring item names
    for i in range(0,len(data_ibs.columns)):
            data_neighbours.ix[i,:10] = data_ibs.ix[0:,i].sort_values(ascending=False)[:10].index
    str = data_neighbours.loc[item][1:].head(5).to_string(index=False)
    list = data_neighbours.loc[item][1:].head(5).tolist()
    str = str.replace('\n',' ')
    str =str.replace('    ','')
    str = str.replace(' ',', ')
    print("Along with",item,", Now You can try these items : \n\n")
    for i in list:
        print(i)




def user_based(userid):
    data["Quantity"] = 1

    itemarray = pd.unique(data.item)
    #This particular view isn't very helpful for us for analysis.
    #This way of data being arranged is called LONG
    #We need it in wide format
    #Converting data from long to wide format
    dataWide = data.pivot("Person", "item", "Quantity")
    #Replace NA with 0
    dataWide.fillna(0, inplace=True)
    #Drop the Person column
```

```python
    data_ib = dataWide.copy()
    data_ib = data_ib.reset_index()
    #Drop the Person column
    #data_ib = data_ib.iloc[:,1:]
    data_ib = data_ib.drop("Person", axis=1)
    # Create a placeholder dataframe listing item vs. item
    data_ibs = pd.DataFrame(index=data_ib.columns, columns=data_ib.columns)
    for i in range(0,len(data_ibs.columns)) :
        # Loop through the columns for each column
        for j in range(0,len(data_ibs.columns)) :
            # Fill in placeholder with cosine similarities
            data_ibs.ix[i,j] = 1-cosine(data_ib.ix[:,i],data_ib.ix[:,j])

    data_neighbours = pd.DataFrame(index=data_ibs.columns,columns=range(1,11))

    # Loop through our similarity dataframe and fill in neighbouring item names
    for i in range(0,len(data_ibs.columns)):
            data_neighbours.ix[i,:10] = data_ibs.ix[0:,i].sort_values(ascending=False)[:10].index
    data_sims1 = dataWide.reset_index()
    # Create a place holder matrix for similarities, and fill in the user name column
    data_sims = pd.DataFrame(index=data_sims1.index,columns=data_sims1.columns)
    data_sims.ix[:,:1] = data_sims1.ix[:,:1]
    data_sims12 = data_sims1.iloc[:50,:]
    data_sims11 = data_sims.iloc[:50,:]
    for i in range(0,len(data_sims11.index)):
        for j in range(1,len(data_sims11.columns)):
            user = data_sims11.index[i]
            product = data_sims11.columns[j]

            if data_sims12.ix[i][j] == 1:
                data_sims11.ix[i][j] = 0
            else:
                product_top_names = data_neighbours.ix[product][1:10]
                product_top_sims = data_ibs.ix[product].sort_values(ascending=False)[1:10]
                user_purchases = data_ib.ix[user,product_top_names]

                #print (i)
                #print (j)

                data_sims11.ix[i][j] = getScore(user_purchases,product_top_sims)
```

```python

    # Get the top products
    data_recommend = pd.DataFrame(index=data_sims.index, columns=['Person','1','2','3','4','5','6'])
    data_recommend.ix[0:,0] = data_sims.ix[:,0]
    # Instead of top product scores, we want to see names
    for i in range(0,len(data_sims.index)):
        data_recommend.ix[i,1:] = data_sims.ix[i,:].sort_values(ascending=False).ix[1:7,].index.transpose()
    # Print a sample
    print("Based on your previous shoppings, these are the items you can buy : \n\n")
    print(data_recommend.ix[userid-1:userid-1,:4].drop("Person",axis=1).to_string(index=False,header=False))
    #print(data_recommend.ix[userid-1:userid-1,:4].drop("Person",axis=1).tolist())

def trending():


    print("Hello, Welcome to ABCD Store\n\n")
    print("Here are some trending products ::\n\n")
    print(data['item'].value_counts().head(5))
```

Listing 2: recomm.py

Let's look at the output now.

```
In [6]: recomm.trending()

Hello, Welcome to ABCD Store


Here are some trending products ::


whole-milk          2513
other-vegetables    1903
rolls/buns          1809
soda                1715
yogurt              1372
Name: item, dtype: int64

In [7]: recomm.item_based("yogurt")

Along with yogurt , Now You can try these items :


whole-milk
other-vegetables
tropical-fruit
rolls/buns
root-vegetables

In [8]: recomm.user_based(3)

Based on your previous shoppings, these are the items you can buy :


cereals  curd  domestic-eggs
```

3.2.3 Feedback Sentiment Analyzer

Customer Feedback Customer feedback is a marketing term that describes the process of obtaining a customer's opinion about a business, product or service.

Customer feedback is so important because it provides marketers and business owners with insight that they can use to improve their business, products and/or overall customer experience.

- It can help improve a product or service
- It offers the best way to measure customer satisfaction
- It can help improve customer retention

Sentiment Analysis Today's algorithm-based sentiment analysis tools can handle huge volumes of customer feedback consistently and accurately. Paired with text analytics, sentiment analysis reveals the customer's opinion about topics ranging from your products and services to your location, your advertisements, or even your competitors.

Sometimes known as "opinion mining", sentiment analysis can let you know if there has been a change in public opinion toward any aspect of your business. Peaks or valleys in sentiment scores give you a place to start if you want to make product improvements, train sales or customer care agents, or create new marketing campaigns.

Implementation Our two algorithms implements sentiment analysis from customer feedback both by using traditional Natural language Processing, and using deep learning with tensorflow. The accuracy of the sentiment analyzer module is state-of-the-art, averaging between 75% to 79%

Languages used Python

Packages used nltk, sklearn, tensorflow, pickle, random, statistics

Code Here is the code for implementing Sentiment Analyzer module using nltk and traditional machine learning algorithms. The first one (preprocess.py) is for preprocessing the training data, the second one (sentiment-mod.py) is for training the data and predicting the sentiment of new customer feedbacks.

```python
# -*- coding: utf-8 -*-
"""
Created on Fri Jul 14 13:56:10 2017

@author: 1399869
"""

import nltk
import random
#from nltk.corpus import movie_reviews
from nltk.classify.scikitlearn import SklearnClassifier
import pickle
from sklearn.naive_bayes import MultinomialNB, BernoulliNB
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.svm import SVC, LinearSVC, NuSVC
from nltk.classify import ClassifierI
from statistics import mode
from nltk.tokenize import word_tokenize



class VoteClassifier(ClassifierI):
    def __init__(self, *classifiers):
        self._classifiers = classifiers

    def classify(self, features):
        votes = []
        for c in self._classifiers:
            v = c.classify(features)
            votes.append(v)
        return mode(votes)

    def confidence(self, features):
        votes = []
        for c in self._classifiers:
            v = c.classify(features)
            votes.append(v)

        choice_votes = votes.count(mode(votes))
        conf = choice_votes / len(votes)
        return conf
```

```
short_pos = open("positive.txt","r").read()
short_neg = open("negative.txt","r").read()

# move this up here
all_words = []
documents = []


#  j is adject, r is adverb, and v is verb
#allowed_word_types = ["J","R","V"]
allowed_word_types = ["J"]

for p in short_pos.split('\n'):
    documents.append( (p, "pos") )
    words = word_tokenize(p)
    pos = nltk.pos_tag(words)
    for w in pos:
        if w[1][0] in allowed_word_types:
            all_words.append(w[0].lower())


for p in short_neg.split('\n'):
    documents.append( (p, "neg") )
    words = word_tokenize(p)
    pos = nltk.pos_tag(words)
    for w in pos:
        if w[1][0] in allowed_word_types:
            all_words.append(w[0].lower())



save_documents = open("pickled_algos/documents.pickle","wb")
pickle.dump(documents, save_documents)
save_documents.close()


all_words = nltk.FreqDist(all_words)


word_features = list(all_words.keys())[:5000]


save_word_features = open("pickled_algos/word_features5k.pickle","wb"
    )
pickle.dump(word_features, save_word_features)
save_word_features.close()


```

```python
def find_features(document):
    words = word_tokenize(document)
    features = {}
    for w in word_features:
        features[w] = (w in words)

    return features

featuresets = [(find_features(rev), category) for (rev, category) in
    documents]
print(featuresets.head(20))
save_featuresets = open("pickled_algos/featuresets.pickle","wb")
pickle.dump(featuresets, save_featuresets)
save_featuresets.close()

random.shuffle(featuresets)
print(len(featuresets))


testing_set = featuresets[10000:]
training_set = featuresets[:10000]


classifier = nltk.NaiveBayesClassifier.train(training_set)
print("Original Naive Bayes Algo accuracy percent:", (nltk.classify.
    accuracy(classifier, testing_set))*100)
classifier.show_most_informative_features(15)

###############
save_classifier = open("pickled_algos/originalnaivebayes5k.pickle","
    wb")
pickle.dump(classifier, save_classifier)
save_classifier.close()

MNB_classifier = SklearnClassifier(MultinomialNB())
MNB_classifier.train(training_set)
print("MNB_classifier accuracy percent:", (nltk.classify.accuracy(
    MNB_classifier, testing_set))*100)

save_classifier = open("pickled_algos/MNB_classifier5k.pickle","wb")
pickle.dump(MNB_classifier, save_classifier)
save_classifier.close()

BernoulliNB_classifier = SklearnClassifier(BernoulliNB())
BernoulliNB_classifier.train(training_set)
print("BernoulliNB_classifier accuracy percent:", (nltk.classify.
    accuracy(BernoulliNB_classifier, testing_set))*100)

```

```python
save_classifier = open("pickled_algos/BernoulliNB_classifier5k.pickle","wb")
pickle.dump(BernoulliNB_classifier, save_classifier)
save_classifier.close()

LogisticRegression_classifier = SklearnClassifier(LogisticRegression())
LogisticRegression_classifier.train(training_set)
print("LogisticRegression_classifier accuracy percent:", (nltk.classify.accuracy(LogisticRegression_classifier, testing_set))*100)

save_classifier = open("pickled_algos/LogisticRegression_classifier5k.pickle","wb")
pickle.dump(LogisticRegression_classifier, save_classifier)
save_classifier.close()


LinearSVC_classifier = SklearnClassifier(LinearSVC())
LinearSVC_classifier.train(training_set)
print("LinearSVC_classifier accuracy percent:", (nltk.classify.accuracy(LinearSVC_classifier, testing_set))*100)

save_classifier = open("pickled_algos/LinearSVC_classifier5k.pickle","wb")
pickle.dump(LinearSVC_classifier, save_classifier)
save_classifier.close()


NuSVC_classifier = SklearnClassifier(NuSVC())
NuSVC_classifier.train(training_set)
print("NuSVC_classifier accuracy percent:", (nltk.classify.accuracy(NuSVC_classifier, testing_set))*100)


SGDC_classifier = SklearnClassifier(SGDClassifier())
SGDC_classifier.train(training_set)
print("SGDClassifier accuracy percent:",nltk.classify.accuracy(SGDC_classifier, testing_set)*100)

save_classifier = open("pickled_algos/SGDC_classifier5k.pickle","wb")
pickle.dump(SGDC_classifier, save_classifier)
save_classifier.close()
```

Listing 3: preprocess.py

```python
# -*- coding: utf-8 -*-
"""
Created on Fri Jul 14 15:49:04 2017

@author: 1399869
"""

#File: sentiment_mod.py

import nltk
import random
#from nltk.corpus import movie_reviews
from nltk.classify.scikitlearn import SklearnClassifier
import pickle
from sklearn.naive_bayes import MultinomialNB, BernoulliNB
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.svm import SVC, LinearSVC, NuSVC
from nltk.classify import ClassifierI
from statistics import mode
from nltk.tokenize import word_tokenize



class VoteClassifier(ClassifierI):
    def __init__(self, *classifiers):
        self._classifiers = classifiers

    def classify(self, features):
        votes = []
        for c in self._classifiers:
            v = c.classify(features)
            votes.append(v)
        return mode(votes)

    def confidence(self, features):
        votes = []
        for c in self._classifiers:
            v = c.classify(features)
            votes.append(v)

        choice_votes = votes.count(mode(votes))
        conf = choice_votes / len(votes)
        return conf


documents_f = open("pickled_algos/documents.pickle", "rb")
documents = pickle.load(documents_f)
documents_f.close()

```

```



word_features5k_f = open("pickled_algos/word_features5k.pickle", "rb")
word_features = pickle.load(word_features5k_f)
word_features5k_f.close()


def find_features(document):
    words = word_tokenize(document)
    features = {}
    for w in word_features:
        features[w] = (w in words)

    return features



featuresets_f = open("pickled_algos/featuresets.pickle", "rb")
featuresets = pickle.load(featuresets_f)
featuresets_f.close()

random.shuffle(featuresets)
print(len(featuresets))

testing_set = featuresets[10000:]
training_set = featuresets[:10000]



open_file = open("pickled_algos/originalnaivebayes5k.pickle", "rb")
classifier = pickle.load(open_file)
open_file.close()


open_file = open("pickled_algos/MNB_classifier5k.pickle", "rb")
MNB_classifier = pickle.load(open_file)
open_file.close()



open_file = open("pickled_algos/BernoulliNB_classifier5k.pickle", "rb")
BernoulliNB_classifier = pickle.load(open_file)
open_file.close()


open_file = open("pickled_algos/LogisticRegression_classifier5k.
```

```
pickle", "rb")
LogisticRegression_classifier = pickle.load(open_file)
open_file.close()


open_file = open("pickled_algos/LinearSVC_classifier5k.pickle", "rb")
LinearSVC_classifier = pickle.load(open_file)
open_file.close()


open_file = open("pickled_algos/SGDC_classifier5k.pickle", "rb")
SGDC_classifier = pickle.load(open_file)
open_file.close()




voted_classifier = VoteClassifier(
                                  classifier,
                                  LinearSVC_classifier,
                                  MNB_classifier,
                                  BernoulliNB_classifier,
                                  LogisticRegression_classifier)




def sentiment(text):
    feats = find_features(text)
    return classifier.classify(feats),classifier.confidence(feats)
```

Listing 4: sentiment-mod.py

To use the code, we have to do the following-

```
import sentiment-mod as s

print(s.sentiment("This app was awesome! The chatbot was great, shopping was wonderful, and there were recommendations..so yea!"))

print(s.sentiment("This app was utter junk. There were absolutely no valuable recommendations. I don't see what the point was at all. Horrible experience, 0/10"))
```

Listing 5: Python example

Code Here is the code for implementing Sentiment Analyzer module using deep learning with tensorflow. The first one (preprocess.py) is for preprocessing the training data, the second one (sentiment-mod.py) is for training the data and predicting the sentiment of new customer feedbacks.

```
# -*- coding: utf-8 -*-
"""
Created on Mon Jul 16 13:04:56 2017

@author: 1399869
"""

import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
import pickle
import numpy as np
import pandas as pd

lemmatizer = WordNetLemmatizer()

'''
polarity 0 = negative. 2 = neutral. 4 = positive.
id
date
query
user
tweet
'''

def init_process(fin,fout):
  outfile = open(fout,'a')
  with open(fin, buffering=200000, encoding='latin-1') as f:
    try:
      for line in f:
        line = line.replace('"','')
        initial_polarity = line.split(',')[0]
        if initial_polarity == '0':
          initial_polarity = [1,0]
        elif initial_polarity == '4':
          initial_polarity = [0,1]

        tweet = line.split(',')[-1]
        outline = str(initial_polarity)+':::'+tweet
        outfile.write(outline)
    except Exception as e:
```

```
      print(str(e))
  outfile.close()

init_process('training.1600000.processed.noemoticon.csv','train_set.csv')
init_process('testdata.manual.2009.06.14.csv','test_set.csv')


def create_lexicon(fin):
  lexicon = []
  with open(fin, 'r', buffering=100000, encoding='latin-1') as f:
    try:
      counter = 1
      content = ''
      for line in f:
        counter += 1
        if (counter/2500.0).is_integer():
          tweet = line.split(':::')[1]
          content += ' '+tweet
          words = word_tokenize(content)
          words = [lemmatizer.lemmatize(i) for i in words]
          lexicon = list(set(lexicon + words))
          print(counter, len(lexicon))

    except Exception as e:
      print(str(e))

  with open('lexicon-2500-2638.pickle','wb') as f:
    pickle.dump(lexicon,f)

create_lexicon('train_set.csv')


def convert_to_vec(fin,fout,lexicon_pickle):
  with open(lexicon_pickle,'rb') as f:
    lexicon = pickle.load(f)
  outfile = open(fout,'a')
  with open(fin, buffering=20000, encoding='latin-1') as f:
    counter = 0
    for line in f:
      counter +=1
      label = line.split(':::')[0]
      tweet = line.split(':::')[1]
      current_words = word_tokenize(tweet.lower())
      current_words = [lemmatizer.lemmatize(i) for i in current_words]

      features = np.zeros(len(lexicon))

```

```
      for word in current_words:
        if word.lower() in lexicon:
          index_value = lexicon.index(word.lower())
          # OR DO +=1, test both
          features[index_value] += 1

      features = list(features)
      outline = str(features)+'::'+str(label)+'\n'
      outfile.write(outline)

    print(counter)

convert_to_vec('test_set.csv','processed-test-set.csv','lexicon-2500-2638.pickle')


def shuffle_data(fin):
  df = pd.read_csv(fin, error_bad_lines=False)
  df = df.iloc[np.random.permutation(len(df))]
  print(df.head())
  df.to_csv('train_set_shuffled.csv', index=False)

shuffle_data('train_set.csv')


def create_test_data_pickle(fin):

  feature_sets = []
  labels = []
  counter = 0
  with open(fin, buffering=20000) as f:
    for line in f:
      try:
        features = list(eval(line.split('::')[0]))
        label = list(eval(line.split('::')[1]))

        feature_sets.append(features)
        labels.append(label)
        counter += 1
      except:
        pass
  print(counter)
  feature_sets = np.array(feature_sets)
  labels = np.array(labels)

create_test_data_pickle('processed-test-set.csv')
```

Listing 6: preprocess-2.py

```python
# -*- coding: utf-8 -*-
"""
Created on Mon Jul 19 13:08:07 2017

@author: 1399869
"""

import tensorflow as tf
import pickle
import numpy as np
import nltk
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()

n_nodes_hl1 = 500
n_nodes_hl2 = 500
n_classes = 2
hm_data = 2000000
batch_size = 32
hm_epochs = 10

x = tf.placeholder('float')
y = tf.placeholder('float')


current_epoch = tf.Variable(1)

hidden_1_layer = {'f_fum':n_nodes_hl1,
                  'weight':tf.Variable(tf.random_normal([2638, n_nodes_hl1])),
                  'bias':tf.Variable(tf.random_normal([n_nodes_hl1]))}

hidden_2_layer = {'f_fum':n_nodes_hl2,
                  'weight':tf.Variable(tf.random_normal([n_nodes_hl1, n_nodes_hl2])),
                  'bias':tf.Variable(tf.random_normal([n_nodes_hl2]))}

output_layer = {'f_fum':None,
                'weight':tf.Variable(tf.random_normal([n_nodes_hl2, n_classes])),
                'bias':tf.Variable(tf.random_normal([n_classes])),}


def neural_network_model(data):

    l1 = tf.add(tf.matmul(data,hidden_1_layer['weight']),
```

```python
hidden_1_layer['bias'])
    l1 = tf.nn.relu(l1)

    l2 = tf.add(tf.matmul(l1,hidden_2_layer['weight']), hidden_2_layer['bias'])
    l2 = tf.nn.relu(l2)

    output = tf.matmul(l2,output_layer['weight']) + output_layer['bias']
    return output

saver = tf.train.Saver()

def use_neural_network(input_data):
    prediction = neural_network_model(x)
    with open('lexicon.pickle','rb') as f:
        lexicon = pickle.load(f)

    with tf.Session() as sess:
        sess.run(tf.initialize_all_variables())
        saver.restore(sess,"model.ckpt")
        current_words = word_tokenize(input_data.lower())
        current_words = [lemmatizer.lemmatize(i) for i in current_words]
        features = np.zeros(len(lexicon))

        for word in current_words:
            if word.lower() in lexicon:
                index_value = lexicon.index(word.lower())
                # OR DO +=1, test both
                features[index_value] += 1

        features = np.array(list(features))
        # pos: [1,0] , argmax: 0
        # neg: [0,1] , argmax: 1
        result = (sess.run(tf.argmax(prediction.eval(feed_dict={x:[features]}),1)))
        if result[0] == 0:
            print('Positive:',input_data)
        elif result[0] == 1:
            print('Negative:',input_data)

use_neural_network("This app was awesome! The chatbot was great")

use_neural_network("This app was  utter  junk, 0/10")
```

Listing 7: sentiment.py

3.2.4 Item Finder Module

In a large retail store, sometimes it gets impossible to find certain item because of huge number of racks in different floors. So, in our app, we implemented an Item-Finder module which will give all the necessary information about the location of any product that customer searches via app.

Language used Python

Packages used Pandas

Code Here is the code of item finder module implementation

```python
# -*- coding: utf-8 -*-
"""
Created on Fri Jun 30 12:39:55 2017

@author: 1399869
"""

import pandas as pd

df = pd.read_csv(r"C:\Users\1399869\Desktop\Products.csv")

#finds an item in the store, returns floor, section, rack
def finder(item):

    print("Your item is located at\n")
    print("Floor No : "+df.loc[df['product_name'] == item]["floor_no"
    ].to_string(index = False))
    print("\nSection : "+df.loc[df['product_name'] == item]["section"
    ].to_string(index = False))
    print("\nRack No : "+df.loc[df['product_name'] == item]["rack_no"
    ].to_string(index = False))
```

Listing 8: itemFinder.py

Let's look at the output now.

```
In [12]: import item_finder as it

In [13]: it.finder("rice")

         Your item is located at

         Floor No : 1

         Section : groceries

         Rack No : 121

In [14]: it.finder("yogurt")

         Your item is located at

         Floor No : 1

         Section : dairy_products

         Rack No : 168

In [16]: it.finder("whisky")

         Your item is located at

         Floor No : 1

         Section : alcohol

         Rack No : 164
```

This module can be imported in any program or Python Interpreter Shell to use the finder function for any product queries.

3.2.5 Real-Time Fastest Checkout Counter

Problem Retail stores deal with a large number of customers every day. The checkout lines are considered to be one of the most critical junctures at a retail outlet. However, retailers are finding it hard to manage these checkout queues effectively. Also, a lot of retailers believe that once a customer enters into checkout line, the sale is almost won. But, that usually isn't the case as customers can still get frustrated and annoyed while waiting in long checkout lines.

The checkout experience creates a lasting impression and effective queue management helps promote customer loyalty and a positive image for the business. Wait times can be prominently displayed for customers to see, inside and outside the store. This is really useful at busy times like lunch breaks when customers are far more likely to enter an apparently busy store if they know in advance, that they will not have to queue for more than a couple of minutes.

Implementation Our implementation uses real-time fastest queue finder. Queue becomes fastest because, during shopping, Customer already added his/her product using the barcode scanner from the app, into his/her virtual cart. After shopping is complete, each cart generates its unique QR Code, which user gives to the cashier. Cashier scans the QR Code and bill is automatically generated. No more waiting in the line for scanning 100s of products one-by-one. Thus the checkout queue becomes really fast.

Language used Python

Packages used numpy, pandas

Code Following is the code for generating fastest checkout queue in real-time when one customer completes his/her shopping.

```python
# -*- coding: utf-8 -*-
"""
Created on Thu Jul  6 12:07:00 2017

@author: 1399869
"""

import pandas as pd
import numpy as np

#arbitrarily taking 3 queues for prototype design
#real-time queues will be passed using command line argument

list1 = [1,3,5,7,9]
list2 = [2,4,6,8,10,12,14]
list3 = [13,17,19,22,25]


df_products = pd.read_csv("Products.csv")
df_user = pd.read_csv("groceries_2.csv")


product1=list()
product2 = list()
product3 = list()


for i in list1:

    product1 +=  df_user.loc[df_user['Person'] == i]["item"].tolist()

for i in list2:

    product2 +=  df_user.loc[df_user['Person'] == i]["item"].tolist()

for i in list3:

    product3 +=  df_user.loc[df_user['Person'] == i]["item"].tolist()

pack_time1 = 0
pack_time2 = 0
pack_time3 = 0

for item in product1:
    pack_time1 += int(df_products.loc[df_products['product_name'] ==
    item]["pack_time"].to_string(index = False))

for item in product2:
```

```
    pack_time2 += int(df_products.loc[df_products['product_name'] ==
   item]["pack_time"].to_string(index = False))


for item in product3:
    pack_time3 += int(df_products.loc[df_products['product_name'] ==
   item]["pack_time"].to_string(index = False))



weight1 = len(product1) * pack_time1
weight2 = len(product2) * pack_time2
weight3 = len(product3) * pack_time3


weight = np.array([weight1,weight2,weight3])

def generate():

    print("\nProducts in the first queue :\n")
    print(product1)

    print("\n\nProducts in the second queue :\n")
    print(product2)

    print("\n\nProducts in the 3rd queue :\n")
    print(product3)


    print("\n\nYour assigned fastest moving checkout queue is: ")
    print("Queue Number "+str(np.argmin(weight)+1))
```
Listing 9: checkout.py

Let's look at the output to see how checkout queue is generated in real-time.

```
In [17]: import checkout

In [18]: checkout.generate()

Products in the first queue :

['citrus-fruit', 'semi-finished-bread', 'margarine', 'ready-soups', 'whole-milk', 'other-vegetables', 'whole-milk', 'condensed-
milk', 'long-life-bakery-product', 'rolls/buns', 'pot-plants']


Products in the second queue :

['tropical-fruit', 'yogurt', 'coffee', 'pip-fruit', 'yogurt', 'cream-cheese-', 'meat-spreads', 'whole-milk', 'butter', 'yogur
t', 'rice', 'abrasive-cleaner', 'other-vegetables', 'UHT-milk', 'rolls/buns', 'bottled-beer', 'liquor-(appetizer)', 'whole-mil
k', 'cereals', 'citrus-fruit', 'tropical-fruit', 'whole-milk', 'butter', 'curd', 'yogurt', 'flour', 'bottled-water', 'dishes',
 'frankfurter', 'rolls/buns', 'soda']


Products in the 3rd queue :

['beef', 'fruit/vegetable-juice', 'chocolate', 'butter-milk', 'pastry', 'tropical-fruit', 'root-vegetables', 'other-vegetable
s', 'frozen-dessert', 'rolls/buns', 'flour', 'sweet-spreads', 'salty-snack', 'waffles', 'candy', 'bathroom-cleaner']


Your assigned fastest moving checkout queue is:
Queue Number 1
```

Use During any session, this method can be called by importing checkout module to assign the fastest queue from real-time queue weights calculated from number of products and packing time, to the customer who just finished his/her shopping and wants to pay the bill.

3.2.6 Interactive Chatbot

Introduction A chatbot (also known as a talkbot, chatterbot, Bot, chatterbox, IM bot, interactive agent, or Artificial Conversational Entity) is a computer program which conducts a conversation via auditory or textual methods.[1] Such programs are often designed to convincingly simulate how a human would behave as a conversational partner, thereby passing the Turing test. Chatbots are typically used in dialog systems for various practical purposes including customer service or information acquisition. Some chatterbots use sophisticated natural language processing systems, but many simpler systems scan for keywords within the input, then pull a reply with the most matching keywords, or the most similar wording pattern, from a database.

Usage Chatbots are often integrated into the dialog systems of, for example, virtual assistants, giving them the ability of, for example, small talking or engaging in casual conversations unrelated to the scopes of their primary expert systems.

- Messaging Platforms
- Apps and websites
- Company internal platforms
- Education

My work In this project, my work was to create a Interactive Chatbot, a personal shopping assistant for the Qfree app, which will help customer in every way possible, just by performimg informal conversations with the end user. Customer will not feel monotonous while shopping and can enjoy this experience, which will in fact, help the company increase customer satisfaction and earn more profits.

Implementation Chatbot can be implemented in various programming languages. many different algorithms can be used to write the proper code. I chose Python over all the languages. I used Natural Language Processing to build the bot, by using NLTK and chatterbot package. For training purpose, I created conversation training data in JSON format and train the chatbot using machine learning approach, specifically using Naive Bayes Classifier.

Languages used Python, JSON

Packages used nltk, chatterbot, os, subprocess

Code Following is the code snippet of chatbot implementation

```
from chatterbot import ChatBot
from chatterbot.trainers import ChatterBotCorpusTrainer
import item_finder
import os
import subprocess
import recomm

def chatbot(user):
    # Create a new instance of a ChatBot
    bot = ChatBot("NOSTAW",silence_performance_warning=True,
    storage_adapter="chatterbot.storage.JsonFileStorageAdapter",
    logic_adapters=[
        "chatterbot.logic.MathematicalEvaluation",

        "chatterbot.logic.BestMatch"
    ],
    input_adapter="chatterbot.input.TerminalAdapter",
    output_adapter="chatterbot.output.TerminalAdapter",
    database="../SecondaryDataBase.json"
    )


    bot.set_trainer(ChatterBotCorpusTrainer)

    # Train the chat bot with the entire english corpus
    bot.train("C:/Users/1399869/Downloads/money.corpus.json",
              "C:/Users/1399869/Downloads/conversations.corpus.json",
              "C:/Users/1399869/Downloads/ai.corpus.json",
              "C:/Users/1399869/Downloads/botprofile.corpus.json",
```

```
                "C:/Users/1399869/Downloads/computers.corpus.json",
        "C:/Users/1399869/Downloads/drugs.corpus.json",
        "C:/Users/1399869/Downloads/greetings.corpus.json",
        "C:/Users/1399869/Downloads/history.corpus.json",
        "C:/Users/1399869/Downloads/humour.corpus.json",
        "C:/Users/1399869/Downloads/literature.corpus.json",
        "C:/Users/1399869/Downloads/movies.corpus.json",
        "C:/Users/1399869/Downloads/politocs.corpus.json",
        "C:/Users/1399869/Downloads/math_words.corpus.json",
        "C:/Users/1399869/Downloads/computers.corpus.json",
        "C:/Users/1399869/Downloads/psychology.corpus.json",
        "C:/Users/1399869/Downloads/science.corpus.json",
        "C:/Users/1399869/Downloads/sports.corpus.json"
        )

    print("Type thoughts to bot.\n")

    # The following loop will execute each time the user enters input
    while True:
        try:
            # We pass None to this method because the parameter
            # is not used by the TerminalAdapter

            bot_input = bot.get_response(None)
            if "finder" in bot_input:
                exec(bot_input)
            elif "trending" in bot_input:
                recomm.trending()
            elif "recommendations" in bot_input:
                recomm.user_based(user)
            print("\n")

        # Press ctrl-c or ctrl-d on the keyboard to exit
        except (KeyboardInterrupt, EOFError, SystemExit):
            break
```

Listing 10: chatbot.py

Output Okay, let's look at the Chatbot to see how it is working. Following is the output -

```
In [1]: import chat1

In [ ]: chat1.chatbot(3)

        Type thoughts to bot.

        Hello
        Hello, how are you today ?


        I am fine
        Okay, tell me how can I help you ?


        Show me some trending items
        Here are some trending items :
        {'whole-milk': 2513, 'soda': 1715, 'yogurt': 1372, 'rolls/buns': 1809, 'other-vegetables': 1903}


        Give me some recommendations
        recommendations for you :
        ['cereals', 'curd', 'domestic-eggs']


        Where is rice /
        item_finder.finder('rice')
        Your item is located at

        Floor No : 1

        Section : groceries

        Rack No : 121
```

Use During any shopping session, whenever the user wants to use the chatbot, Chatbot module will be imported and according to the user's userid, chatbot() function will be called to start the chatbot. It will a virtual assistant right to the end of his/her shopping.

3.2.7 Integration Module

Implementation This is a module which is used to take input data from Node.JS code as JSON format and give output back to the Node.JS code, which will be sent to the front end of the application. This integration module can be used to call any of the previously described modules according to the input data, to execute the intended python module.

Languages used Python

Packages used ast, recomm, itemfinder, sentiment, checkout

Code Following is the implementation of Python Integration Module.

```
import recomm1
import ast
while(True):

  a = input("give input : ")
  d = ast.literal_eval(a)

  try:

    if(d['t']):

      recomm1.trending()


  except:
    pass

  try:
    if(d['u']):

      recomm1.user_based(d['u'])

  except:
    pass

  try:
```

```

    if(d['i']):

      recomm1.item_based(d['i'])

  except:
    pass
  try:

    if(d['s']):
      import sentiment1
      sentiment1.analyzer(d['s'])

  except:
    pass

  try:

    if(d['c']):
      import checkout
      checkout.generate()

  except:
    pass

  try:

    if(d['b']):
      import chat1
      chat1.chatbot(d['b'])

  except:
    pass

  try:

    if(d['f']):
      import item_finder
      item_finder.finder(d['f'])

  except:
    pass
```

Listing 11: integration.py

Output Okay, let's look at the output of the integration module to see how it is working. Following is the output -

```
In [*]: import test2

        give input : {"i":"yogurt"}
        ['whole-milk', 'other-vegetables', 'tropical-fruit', 'rolls/buns', 'root-vegetables']
        give input : {"u":5}
        ['coffee', 'cereals', 'chocolate']
        give input : {"t":1}
        {'yogurt': 1372, 'soda': 1715, 'whole-milk': 2513, 'rolls/buns': 1809, 'other-vegetables': 1903}
        give input : {"c":5}
        2
        give input : {"s":"This app works very good, awesome experience"}
        {'Probability': 0.85, 'Predicted sentiment': 'Positive'}
        give input : {"b":4}
        Type thoughts to bot.

        Hey
        Hello, how are you today ?


        I am fine.
        Okay, tell me how can I help you ?


        Tell me where is rice
        item_finder.finder('rice')
        Your item is located at

        Floor No : 1

        Section : groceries

        Rack No : 121
```

Use During any shopping session, any request from the front end of the app will go as a JSON input through Node.JS to this python module, it will execute the required module and send the output to the node.js code, which will send the data to the front end of the app.

4 Scope for Improvement

These systems can be implemented later for improvement.

- Create a Machine Vision and Image Processing module which will track the real-time checkout queue traffic. It will also automate the whole process of billing.
- Add RFID tags to each of the products and RFID scanner to automate the process of scanning and updating the database whenever a product goes into the cart of a customer or moves back to the racks.
- Deploy Beacons in aisles to create Proximity-Sensing Notification System, which will give notifications of offers and food-calorie values of different products whenever a customer moves towards that specific aisle, which contains those products.
- Use of Touch/Capacitive/Pressure Sensors in the racks to identify if any product is taken from it or moved back into the rack, then RFID tags to identify the specific product. It will automate the whole process of cart system.
- Ultra-Wide-Band (UWB) localization system can be implemented to measure distance from the aisles and location of the customer, better than narrowband radio systems like Wifi and Bluetooth.
- Use of third-party Image Recognition tools like Vuforia, clarifAI, Watson to automate checkout queue traffic system and fastest checkout counter selection.

5 Conclusion

This is a prototype version of an application, which can revolutionize Retail Industry. It will definitely make Retail Store Shopping hassle-free, and take customer satisfaction to a whole new level. Revenue and profits will be sky-high. It will benefit both the company and end-users in many different ways.

Automation and Machine Intelligence is the new buzz in the industry for the last 5 years, and it will dominate the industry for the next century and more. So, reinventing markets by using Artificial Intelligence and Machine Learning techniques will be a stepping stone to this AI era for the Retail Companies.