```
from flask import Flask, render_template, request, jsonify
import atexit, os, json, nltk
import pandas as pd
import numpy as np
import nltk
#nltk.download()

import cf_deployment_tracker
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from textblob import TextBlob

from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from nltk.corpus import stopwords

import sys, fuzzy, re, codecs, unicodedata

############################################################
# Initialize program variables
app = Flask(__name__, static_url_path='/static')

# On Bluemix, get the port number from the environment variable PORT
# When running this app on the local machine, default the port to 8000
port = int(os.getenv('PORT', 8000))

soundex = fuzzy.Soundex(4)
EMOTIWORDS = ["awesome", "great", "love", "hate", "just", "because", "museum", "contacted", "don't
know", "does not", "you", "special"]
FEATURES   = ["battery", "screen", "display", "processor", "cpu", "memory", "price", "touch", "camera"]
SENTIMENTS = ["elegant", "thin", "beast"]
DATA = ''
CLASSIFIER = ''
#HOME_DIR = re.sub(r'\\', r'\\\\', os.getcwd())

def clean_text_stopwords(line):
  text = line.lower()
  text = re.sub(r'\.+', '.', text)
  text = unicodedata.normalize('NFKD', text).encode('ascii','ignore')
  text = text.split(' ')
  stops = set(stopwords.words("english"))
  text = [w for w in text if not w in stops]
  text = ' '.join(text)
  return text

def get_summary(review):
  rev_sum = {}
  for line in review.split('.'):
    senti = get_sentiment(line)
    for elem in get_features(line):
      elem = elem.encode('ascii', 'ignore')
      if elem not in rev_sum or rev_sum[elem] < senti:
        rev_sum[elem] = senti
  return rev_sum


def get_features(line):
  features = []
  tokens = [w for w in line.split(' ') if w in FEATURES]
  if len(tokens) > 0:
    features.extend(tokens)
  else:
    features.extend(CLASSIFIER.predict([line]))
  return features


def clean_text(line):
  temp = []
  for word in line.split(' '):
    word = re.sub(r'\.+', '.', word)
    #word = unicodedata.normalize('NFKD', word).encode('ascii','ignore')
    for emo in EMOTIWORDS:
      if soundex(word) == soundex(emo):
        word = emo
    temp.append(word)
  clean_line = ' '.join(temp)
  return clean_line


def clean_line(line):
  '''
  Utility function to clean text by removing links, special characters
  using simple regex statements.
  '''
  return ' '.join(re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(\w+:\/\/\S+)", " ", line).split())


def get_sentiment(line):
  '''
  Utility function to classify sentiment of passed line
  using textblob's sentiment method
  '''
  # create TextBlob object of passed tweet text
  analysis = TextBlob(clean_line(line))
  return analysis.sentiment.polarity

############################################################
# import the data to be summarised, it would be a JSON file
# reviewFile = 'reviews.json'
# reviewIndx = 3
#
# revd = pd.read_json(reviewFile)
# np_arr = revd.as_matrix()
# target = [clean_text_stopwords(w) for w in np_arr[:,reviewIndx]]
# target = [clean_line(w) for w in np_arr[:,reviewIndx]]

############################################################
# train our model
def train_model():
  trainFile = 'reviews_mod.csv'
  trainCol  = 'review-comment'
  data = pd.read_csv(trainFile)
  for index, line in data.iterrows():
    text = clean_text(data.iloc[index][trainCol])
    data.iloc[index][trainCol] = text
  numpy_array = data.as_matrix()
  X = numpy_array[:,0]
  Y = numpy_array[:,1]
  # This has been the best method for classification
  text_clf = Pipeline([('vect', CountVectorizer(stop_words='english', ngram_range=(1, 2))), ('clf',
MultinomialNB())])
  classifier = text_clf.fit(X,Y)
  return data, text_clf

@app.route('/refresh_model', methods=['POST'])
def refresh_model():
    DATA, CLASSIFIER = train_model()
    print("COMPLETED REFRESH !!")
    return 'OK'

# define app routes
@app.route('/')
def home():
    DATA, CLASSIFIER = train_model()
    data_json = DATA.to_dict('records')
    return render_template('review.html', data_json=data_json)

@app.route('/evaluate_review')
@app.route('/evaluate_review/')
def evaluate_review():
  # form parameters
  reviewText = request.args.get('reviewText')
  obj = get_summary(reviewText)
  data_json = DATA.to_dict('records')
  return render_template('review.html', reviewText=reviewText, obj=obj, data_json=data_json)

if __name__ == '__main__':
  DATA, CLASSIFIER = train_model()
  app.run(host='0.0.0.0', port=port, debug=True)
```