```
import pandas as pd import numpy as np import re # nltk.download('stopwords') from nltk.corpus import stopwords stop_words=stopwords.words('english') from nltk.stem import WordNetLemmatizer import nltk import spacy from imblearn.over_sampling import SMOTE from sklearn.model_selection import train_test_split from sklearn.feature_extraction.text import TfidfVectorizer from sklearn.decomposition import PCA from sklearn.svm import SVC from pickle import dump from pickle import load from sklearn.linear_model import LogisticRegression import streamlit as st from sklearn.feature_extraction.text import TfidfTransformer from scipy.sparse import coo_matrix from spacy.lang.en import English # import en_core_web_sm # spacy.load("en_core_web_sm") #Lemmatization wordnet=WordNetLemmatizer() #Stop word stop_words=stopwords.words('english') nlp=spacy.load("en_core_web_sm") # Varibale created for words which are not included in the stopwords not_stopwords = ("aren", "aren't", "couldn", "couldn't", "didn", "didn't", "doesn", "doesn't", "don", "don't", "hadn", "hadn't", "hasn", "hasn't", "haven", "haven't", "isn", "isn't", "mustn", "mustn't", "no", "not", "only", "shouldn", "shouldn't", "should've", "wasn", "wasn't", "weren", "weren't", "will", "wouldn", "wouldn't", "won't", "very") stop_words_ = [words for words in stop_words if words not in not_stopwords] # Additional words added in the stop word list stop_words_.append("I") stop_words_.append("the") stop_words_.append("s") # Stop word for keyword extraction stop_words_keywords = stopwords.words('english') # special additioanl stop words added for keyword extraction stop_words_keywords.extend([ "will", "always", "go", "one", "very", "good", "only", "mr", "lot", "two", "th", "etc", "don", "due", "didn", "since", "nt", "ms", "ok", "almost", "put", "pm", "hyatt", "grand", "till", "add", "let", "hotel", "able", "per", "st", "couldn", "yet", "par", "hi", "well", "would", "I", "the", "s", "also", "great", "get", "like", "take", "thank" ]) def Prediction(corpus): output=[] #convert to string review =str(corpus) #to handle punctuations review = re.sub('[^a-zA-Z]', ' ', review) # Converting Text to Lower case review = review.lower() # Spliting each words - eg ['I','was','happy'] review = review.split() # Applying Lemmitization for the words eg: Argument -> Argue - Using Spacy Library review = nlp(' '.join(review)) review = [token.lemma_ for token in review] # Removal of stop words review = [word for word in review if word not in stop_words_] # Joining the words in sentences review = ' '.join(review) output.append(review) # TFIDF -Pickel file loaded_TFIDF = load(open('model_TFIDF.sav', 'rb')) #converted to number by TFIDF X=pd.DataFrame((loaded_TFIDF.transform(output)).toarray()) # PCA pickle File #loaded_pca= load(open('pca.sav','rb')) # apply PCA #X_PCA= loaded_pca.transform(X) #model pickle file loaded_model= load(open('finalized_model.sav','rb')) #precition and converted to integer pred = int(loaded_model.predict(X)) if pred==1: return 'Positive' else: return 'Negative' def keywords(corpus): output2=[] #convert to string review =str(corpus) #to handle punctuations review = re.sub('[^a-zA-Z]', ' ', review) # Converting Text to Lower case review = review.lower() # Spliting each words - eg ['I','was','happy'] review = review.split() # Applying Lemmitization for the words eg: Argument -> Argue - Using Spacy Library review = nlp(' '.join(review)) review = [token.lemma_ for token in review] # Removal of stop words review = [word for word in review if word not in stop_words_keywords] # Joining the words in sentences review = ' '.join(review) output2.append(review) tfidf2 = TfidfVectorizer(norm="l2",analyzer='word', stop_words=stop_words_keywords,ngram_range=(1,2)) tfidf2_x = tfidf2.fit_transform(output2) tfidf_transformer = TfidfTransformer(smooth_idf=True, use_idf=True) tfidf_transformer.fit(tfidf2_x) # get feature names feature_names = tfidf2.get_feature_names() # generate tf-idf for the given document tf_idf_vector = tfidf_transformer.transform(tfidf2.transform(output2)) def sort_coo(coo_matrix): tuples = zip(coo_matrix.col, coo_matrix.data) return sorted(tuples, key=lambda x: (x[1], x[0]), reverse=True) #sort the tf-idf vectors by descending order of scores sorted_items=sort_coo(tf_idf_vector.tocoo()) #extract only the top n, n here is 10 def extract_topn_from_vector(feature_names, sorted_items, topn=10): """get the feature names and tf-idf score of top n items""" #use only topn items from vector sorted_items = sorted_items[:topn] score_vals = [] feature_vals = [] # word index and corresponding tf-idf score for idx, score in sorted_items: #keep track of feature name and its corresponding score score_vals.append(round(score, 3)) feature_vals.append(feature_names[idx]) #create a tuples of feature,score #results = zip(feature_vals,score_vals) results= feature_vals return pd.Series (results) attributes=extract_topn_from_vector(feature_names,sorted_items,10) return attributes
```

this is the main function in which we define our webpage def main(): # front end elements of the web page html_temp = """

Sentiment Analysis for Hotel Review

""" # display the front end aspect st.markdown(html_temp, unsafe_allow_html=True) # following lines create boxes in which user can enter data required to make prediction # Textbox for text user is entering st.subheader("Enter the text you'd like to analyze.") text = st.text_input('Enter text') # text is stored in this variable # when 'Button' is clicked, make the prediction and store it if st.button("Predict"): predict = Prediction(text) st.success('The Sentiment of the review is {}'.format(predict)) #if st.button("IMP Attributes"): st.subheader("Important Attributes in Reviews") imp_att=keywords(text) for i in imp_att: st.success(i) if __name__ == '__main__': main()