Intention Recognition in Chinese based on SVM

we will be given a sentence and each sentence correspondes to a specific label for its theme, like weather, music or stock. We would then create model on training set to recognize its intent by analyzing the sentences and then fit the model on testing set.

> 1. read the file and show the data, use indexes to represent the labels
> 2. cut the sentences
> 3. eliminate non-informative words
> 4. count words and eliminate rarely used words.
> 5. use `tfidf`
> 6. use `tiny-word-embedding`
> 7. create Linear and Non-Linear SVM models based on these two methods

In [4]:
```
import numpy as np
import pandas as pd
import jieba
import requests
import os
from collections import Counter
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer
from gensim.models import Word2Vec
import text2vec
from sklearn.svm import SVC
from sklearn import metrics
import pickle
import warnings

warnings.filterwarnings('ignore')
```

```
/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/ge
nsim/similarities/__init__.py:15: UserWarning: The gensim.similarities.levenshte
in submodule is disabled, because the optional Levenshtein package <https://pyp
i.org/project/python-Levenshtein/> is unavailable. Install Levenhstein (e.g. `pi
p install python-Levenshtein`) to suppress this warning.
  warnings.warn(msg)
```

In [5]:
```
if not os.path.exists('train.json'):
    trainData = requests.get("https://worksheets.codalab.org/rest/bundles/0x0161
    with open("train.json", "wb") as f:
        f.write(trainData.content)

if not os.path.exists('test.json'):
    testData = requests.get("https://worksheets.codalab.org/rest/bundles/0x1f96b
    with open("test.json", "wb") as f:
        f.write(testData.content)
```

In [6]:
```
#read data to new dataframes
train_df = pd.read_json("train.json").transpose()
test_df = pd.read_json("test.json").transpose()
```

In [7]:
```
# look the data
print ("training and testing data", train_df.shape, test_df.shape)
print ("unique labels", train_df.label.unique())
train_df.head()
```

```
training and testing data (2299, 2) (770, 2)
unique labels ['weather' 'map' 'cookbook' 'health' 'chat' 'train' 'calc' 'transl
ation'
 'music' 'tvchannel' 'poetry' 'telephone' 'stock' 'radio' 'contacts'
 'lottery' 'website' 'video' 'news' 'bus' 'app' 'flight' 'epg' 'message'
 'match' 'schedule' 'novel' 'riddle' 'email' 'datetime' 'cinemas']
```

Out[7]:

	query	label
0	今天东莞天气如何	weather
1	从观音桥到重庆市图书馆怎么走	map
2	鸭蛋怎么腌?	cookbook
3	怎么治疗牛皮癣	health
4	唠什么	chat

In [8]:
```
labelName = train_df.label.unique()

# map label and index using zip dict
label_dict=dict(zip(labelName,range(len(labelName))))
```

In [9]:
```
# look label and index
index_dict=dict(zip([str(i) for i in range(len(labelName))],labelName))
```

In [10]:
```
# count the number of each label using groupby().count() / value_counts()
train_df.label.value_counts()
```

Out[10]:
```
chat           455
cookbook       269
video          182
epg            107
poetry         102
tvchannel       71
stock           71
train           70
map             68
weather         66
music           66
telephone       63
message         63
flight          62
translation     61
news            58
health          55
website         54
app             53
riddle          34
contacts        30
schedule        29
match           24
bus             24
radio           24
```

```
lottery         24
email           24
calc            24
cinemas         24
novel           24
datetime        18
Name: label, dtype: int64
```

In [11]:
```
# translate label into index
train_df["labelIndex"] = train_df.label.map(label_dict)# TODO
test_df["labelIndex"] = test_df.label.map(label_dict)# TODO
```

In [12]:
```
# use jieba.cut to divide the sentence
import re
def query_cut(query):
    tokens = list(jieba.cut(query,cut_all=False))
    return tokens
train_df["queryCut"] = train_df["query"].apply(query_cut)
test_df["queryCut"] = test_df["query"].apply(query_cut)
```

In [13]:
```
# read the stop words to eliminate
with open("stopwords.txt","r", encoding='utf-8') as f:
    stopWords=f.read().split('\n')
```

In [14]:
```
# use stop words to filter the result

def rm_stop_word(wordList):
    valid_words=[]
    for word in wordList:
        if word not in stopWords:
            valid_words.append(word)
    return valid_words

train_df["queryCutRMStopWord"] = train_df["queryCut"].apply(rm_stop_word)
test_df["queryCutRMStopWord"] = test_df["queryCut"].apply(rm_stop_word)
```

In [15]:
```
# count the number of words used by collections.Counter()
allWords = [word for query in train_df.queryCutRMStopWord for word in query]
freWord =  Counter(allWords)
```

In [16]:
```
# filter the number of words used less than 3
highFreWords = [word for word in freWord.keys() if freWord[word]>3] #larger than
def rm_low_fre_word(query):
    return [word for word in query if word in highFreWords]

train_df["queryFinal"] = train_df["queryCutRMStopWord"].apply(rm_low_fre_word)
test_df["queryFinal"] = test_df["queryCutRMStopWord"].apply(rm_low_fre_word)
```

In [17]:
```
# transform the word list into tfidf form
trainText = [' '.join(query) for query in train_df["queryFinal"]]
testText = [' '.join(query) for query in test_df["queryFinal"]]
allText = trainText+testText
```

```
# sklearn tfidf vector fit_transform
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer=TfidfVectorizer()
tfidf=vectorizer.fit_transform(allText)
```

In [18]:
```
# create train and test set using sklearn train_test_split()
trainLen = len(train_df)
train_x_tfidf = tfidf.toarray()[0:trainLen]
test_x_tfidf = tfidf.toarray()[trainLen:]
train_y_tfidf = train_df["labelIndex"]
test_y_tfidf = test_df["labelIndex"]
```

In [19]:
```
# shape of the result
print("train_x_tfidf.shape =",train_x_tfidf.shape)
print("train_y_tfidf.shape =",train_y_tfidf.shape)
print("test_x_tfidf.shape =",test_x_tfidf.shape)
print("test_y_tfidf.shape =",test_y_tfidf.shape)
```

```
train_x_tfidf.shape = (2299, 238)
train_y_tfidf.shape = (2299,)
test_x_tfidf.shape = (770, 238)
test_y_tfidf.shape = (770,)
```

In [20]:
```
# read the embedding
with open("tiny_word2vec.pickle","rb") as f:
    word2vec = pickle.load(f)
```

In [21]:
```
#read the package
with open("tiny_word2vec.pickle","rb") as f:
    word2vec = pickle.load(f)

# have filtered words into higher-order vectors
vocabulary = word2vec.keys()
def sentence2vec(query):
    sentence = []
    for word in query:
        if word in word2vec:
            sentence.append(word2vec[word])
    sentence = np.array(sentence)
    if len(sentence)>0:
        sentence= sentence.sum(axis=0)/len(sentence)
    else:
        sentence = np.zeros(shape=(300,))
    return sentence
```

In [22]:
```
# transform the sentence into vector forms
train_x_vec = np.vstack(train_df["queryCutRMStopWord"].apply(sentence2vec))
test_x_vec = np.vstack(test_df["queryCutRMStopWord"].apply(sentence2vec))
train_y_vec = train_df["labelIndex"]
test_y_vec = test_df["labelIndex"]
```

In [23]:
```
# look into the data
print("train_x_vec.shape =",train_x_vec.shape)
```

```
print("train_y_vec.shape =",train_y_vec.shape)
print("test_x_vec.shape =",test_x_vec.shape)
print("test_y_vec.shape =",test_y_vec.shape)
```

```
train_x_vec.shape = (2299, 300)
train_y_vec.shape = (2299,)
test_x_vec.shape = (770, 300)
test_y_vec.shape = (770,)
```

In [24]:

```
# use tfidf to construct linear model  hint: SVC()
from sklearn.svm import SVC
tfidfLinearSVM=SVC(C=1,kernel='linear',decision_function_shape='ovr')
tfidfLinearSVM.fit(train_x_tfidf,train_y_tfidf)
# get model result,  accuracy,  F1_score

print('train accuracy %s' % metrics.accuracy_score(train_y_tfidf, tfidfLinearSVM
print('train F1_score %s' % metrics.f1_score(train_y_tfidf, tfidfLinearSVM.predi
print('test accuracy %s' % metrics.accuracy_score(test_y_tfidf, tfidfLinearSVM.p
print('test F1_score %s' % metrics.f1_score(test_y_tfidf, tfidfLinearSVM.predict
```

```
train accuracy 0.727707698999565
train F1_score 0.7708505943829783
test accuracy 0.6831168831168831
test F1_score 0.6954557599168526
```

In [25]:

```
# use tfidf to construct `rbf` SVM  model which is non-linear
tfidfKernelizedSVM=SVC(C=1,kernel='rbf',decision_function_shape='ovr')
tfidfKernelizedSVM.fit(train_x_tfidf,train_y_tfidf)

# get model result,  accuracy,  F1_score

print('train accuracy %s' % metrics.accuracy_score(train_y_tfidf, tfidfKernelize
print('train F1_score %s' % metrics.f1_score(train_y_tfidf, tfidfKernelizedSVM.p
print('test accuracy %s' % metrics.accuracy_score(test_y_tfidf, tfidfKernelizedS
print('test F1_score %s' % metrics.f1_score(test_y_tfidf, tfidfKernelizedSVM.pre
```

```
train accuracy 0.745541539799913
train F1_score 0.7984100288786787
test accuracy 0.6883116883116883
test F1_score 0.7020022181879286
```

In [26]:

```
# use embedding and construct linear SVM model
word2vecLinearSVM=SVC(C=1,kernel='linear',decision_function_shape='ovr')
word2vecLinearSVM.fit(train_x_vec,train_y_vec)

# get model result,  accuracy,  F1_score

print('train accuracy %s' % metrics.accuracy_score(train_y_vec, word2vecLinearSV
print('train F1_score %s' % metrics.f1_score(train_y_vec, word2vecLinearSVM.pred
print('test accuracy %s' % metrics.accuracy_score(test_y_vec, word2vecLinearSVM.
print('test F1_score %s' % metrics.f1_score(test_y_vec, word2vecLinearSVM.predic
```

```
train accuracy 0.9778164419312745
train F1_score 0.9818051685775911
test accuracy 0.8467532467532467
test F1_score 0.8565368406833987
```

In [27]:

```
# use embedding and construct `rbf` SVM model which is non-linear
word2vecKernelizedSVM=SVC(C=1,kernel='rbf',decision_function_shape='ovr')
```

```
word2vecKernelizedSVM.fit(train_x_vec,train_y_vec)


# get model result: accuracy,  F1_score
print('train accuracy %s' % metrics.accuracy_score(train_y_vec, word2vecKerneliz
print('train F1_score %s' % metrics.f1_score(train_y_vec, word2vecKernelizedSVM.
print('test accuracy %s' % metrics.accuracy_score(test_y_vec, word2vecKernelized
print('test F1_score %s' % metrics.f1_score(test_y_vec, word2vecKernelizedSVM.pr
```

```
train accuracy 0.9386689865158765
train F1_score 0.9353701170287393
test accuracy 0.8428571428571429
test F1_score 0.840815869158248
```

By looking into the result, it is shown that converting words into vectors form is much better than transforming them into tfidf forms. Linear SVM is slightly better.