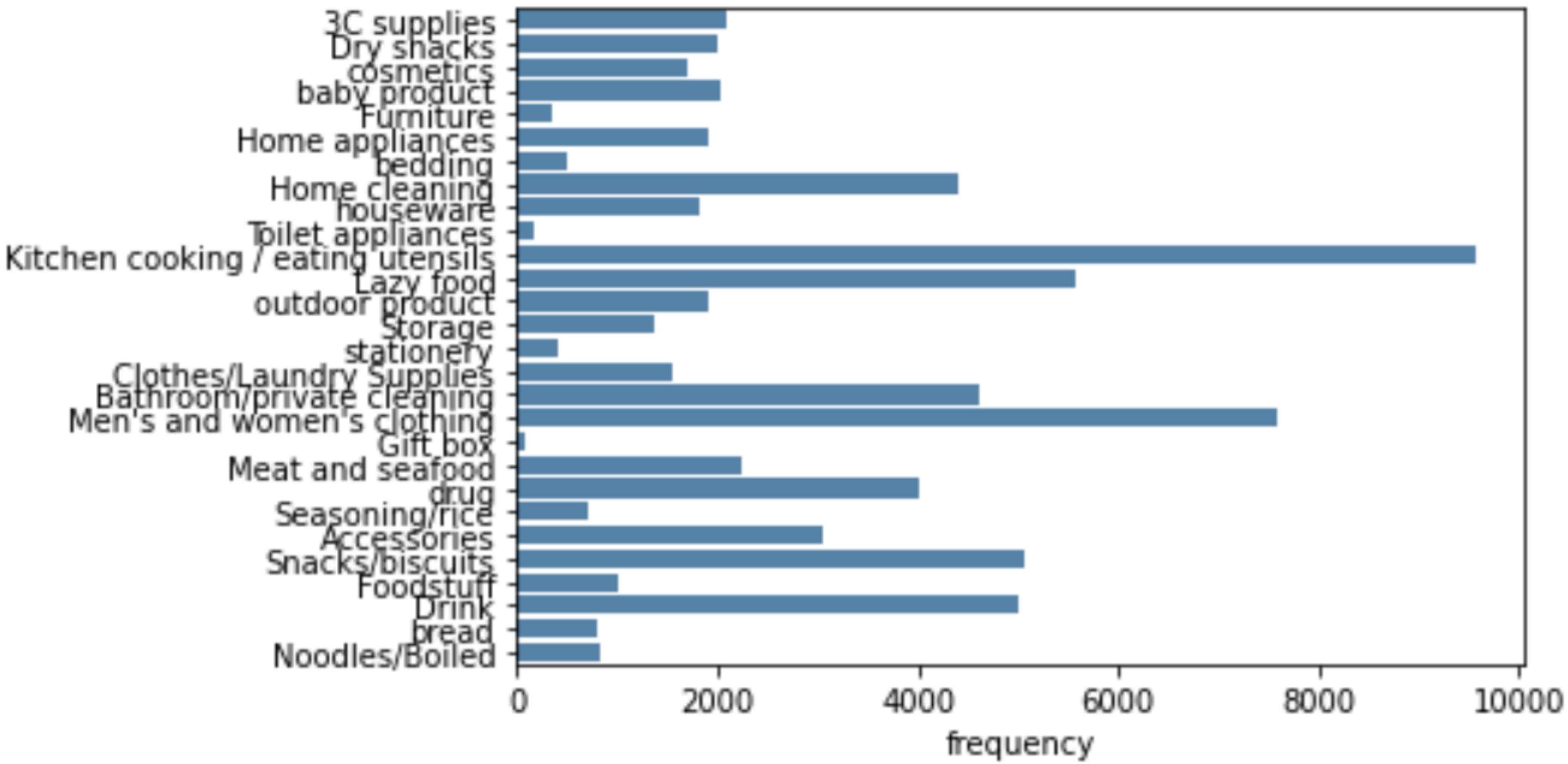
發票分類方法及結果

目錄

- 專案簡介
- 使用資料
- 資料篩選(1)
- 資料分布
- 資料篩選(2)
- 分析前處理
- 計分系統
- 使用模型
- 程式碼連結

專案簡介

資料分布(極度不平衡)


3C supplies
Dry snacks
cosmetics
baby product
Furniture
Home appliances
bedding
Home cleaning
houseware
Toilet appliances
Kitchen cooking / eating utensils
Lazy food
outdoor product
Storage
stationery
Clothes/Laundry Supplies
Bathroom/private cleaning
Men's and women's clothing
Gift box
Meat and seafood
drug
Seasoning/rice
Accessories
Snacks/biscuits
Foodstuff
Drink
bread
Noodles/Boiled
0
2000
4000
6000
8000
10000
frequency

Training set 與 Testing set

- 採 8:2 進行抽樣，每次抽樣前會進行隨機排序(非按類別進行抽樣)

- 全為真實資料，並沒有利用資料擾動來增加新資料

- 樣本數(testing set結果)服從常態分配(Kolmogorov–Smirnov Test)(母體非常態分配)，因此testing set樣本數夠多

採用模型:貝氏分群 + TF-IDF計分系統

```python
from collections import Counter

count = 0
count_rem_w = 0
count_rem = 0
count_rem_1 = 0
count_rem_n = 0
for i in range(len(test)):
    if i % 100 ==0:
        print(i/len(test))
    word = str(test.loc[i,"spt_data"])
    #print(word)
    words = word.split("/")
    words = delete_unnecessary_word(words)
    words = [x for x in words if str(x) != 'nan']
    guess = []
    for j in range(len(words)):
        #print(words[j])
        if words[j] == "麵包":
            guess = ["麵包"]
            break
        elif words[j] == "禮盒":
            guess = ["禮盒"]
            break
        elif words[j] == "童" or words[j] == "女童":
            guess = ["嬰兒用品"]
            break
        guess.append(classifier.classify(word_feats(words[j])))
    #print(guess, "ans =" ,test.loc[i,"y"], test.loc[i,"x"])
    ans = new_score_alg(words, guess, 0)
    if len(rem_new):
        count_rem = count_rem + 1

    if test.loc[i,"cate"]==ans:
        count = count + 1
    else:
        if len(rem_new):
            count_rem_w = count_rem_w + 1
            if len(rem_new) == 1:
                count_rem_1 = count_rem_1 + 1
            else:
                count_rem_n = count_rem_n + 1
        print("guess = ",ans," ","correct = ",test.loc[i,"cate"])
        wrong_pair(test.loc[i,"cate"] ,ans)
        print(words)
    rem_new = []
print("多元判斷的機率 = ", count_rem / len(test))
print("多數決情況成功機率",(count - count_rem + count_rem_w) / (len(test) - count_rem))
print("在錯誤情況下多元判斷失敗機率 = ", count_rem_w / (len(test) - count))
print("多元判斷錯誤下一，為一元之機率 = ", count_rem_1 / count_rem_w)
print(count/len(test))
```

專案簡介

最終預測結果

- 多元判斷的機率 = 0.12067178104913955
- 利用貝氏統計模型 0.8770730173701171
- 在錯誤情況下多元判斷失敗機率 = 0.36111111111111111
- 多元判斷錯誤下一，為一元之機率 = 1.0
- 預測率: 0.8308106987352271

使用資料

商品資料

- 愛買商品資料
- 7-11商品資料
- 大潤發商品資料
- 家樂福發票資料

字典

- Dict.txt.big
- 宿舍物品勾選清單
- 顏色表
- 國家名稱
- 公司名稱

其他資料

- 各類青菜資料
- 不同廠商品項對應資料

資料篩選(1)

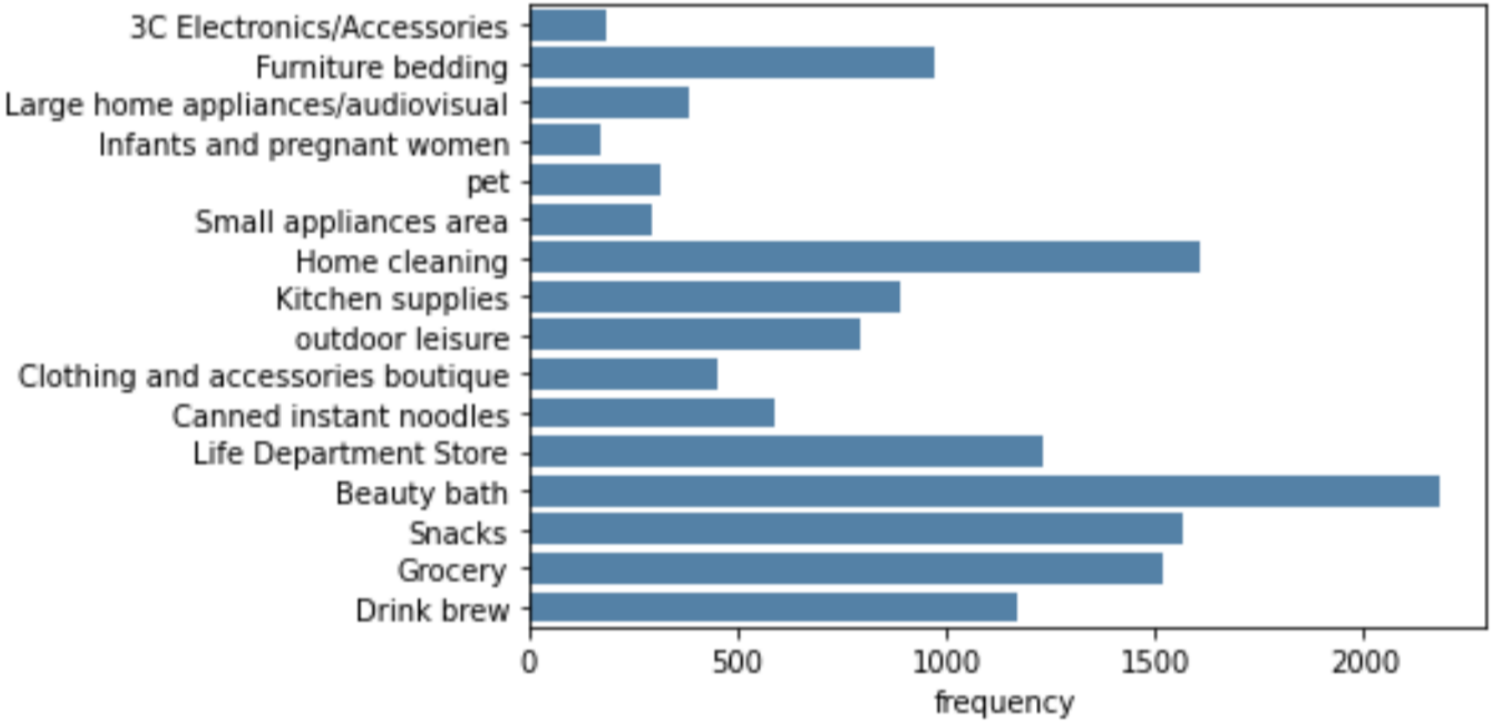
資料篩選(1)

- 資料問題：有些商品類別過少、無分類或不具意義之分類

- 解決方式：
  1. 將類別商品數量 < 10 者刪去
  2. 將特價商品區、免運商品區...刪除，以處理同類商品出現在不同類別等問題。

資料分布

資料分布圖(僅包含愛買資料)



資料問題及處理方法

資料問題

- 就整體資料而言，資料偏誤較大(少數類別商品數量極少)
- 美妝產品和清潔用品相對較高

處理方法：不處理

1. 貝氏統計較不會大幅度受資料偏誤影響。
2. 大多數賣場商品分布相似，故可假設此資料偏誤為一種適用於賣場的分配

資料處理與篩選

Google translate

方法:

1. 利用 `from googletrans import Translator`
   來進行翻譯(以python 完成)
2. 利用google線上翻譯(可直接翻譯整份CSV)

採用方法:使用google線上翻譯

1. 雖然用程式翻譯可以減少人力操作，但在大型資料的情況下，使用程式翻譯的時間遠大於線上翻譯

中文Jeiba斷詞字典

- dict.txt.big.txt（網路公開字典）
- 宿舍物品勾選清單（日常用品）
- name_data（公司名稱）
- country_name（國家名稱）
- 顏色表
- ["天地合補","家樂氏","萬歲牌","伯朗","馬玉山","白蘭氏","麥斯威爾","天仁","桂格 “,"拿鐵","抹茶","杜老爺","依必朗","薇薇特南果","喜瑞爾"]

中文資料篩選

分析前基本處理:

1. 將括號及括號中的字去除
2. 去除標點符號

中文分類模型方法:

方法1: 僅作基本處理

方法2:

- 將無法作為判斷的字詞刪去
- 將字數 < 2的詞刪去

方法3:

- 方法2 + 只取名詞

將無法作為判斷的字詞刪去

STEP 1：把training set放入模型中進行預測，將預測錯誤的品項單詞儲存並計數，並輸出CSV。

STEP 2：手動挑選錯誤率高的單詞，並存入新的CSV中。

STEP 3：在預測時將錯誤率高的詞彙刪除，提高預測率。

將字數 < 2的詞刪去

理由:

- 單一中文字通常不具意義，如「和」、「片」、「個」
- 在資料中數量過多，導致資料預測效果不佳。
- 可大幅增加運算效率

只取名詞

```
#collection = ['r' , 'nr' , 'rn' , 'n' , 'n']
def fuzzyfinder(user_input, collection):
    suggestions = []
    pattern = '.*'.join(user_input) # Converts 'djm' to 'd.*j.*m'
    regex = re.compile(pattern)     # Compiles a regex.
    for item in collection:
        match = regex.search(item)  # Checks if the current item matches the regex.
        if match:
            suggestions.append(item)
    return suggestions
```

結果：效果不佳

推測可能問題：

- Jeiba套件對於詞性的預測能力有限，且此資料為商品名稱資料，較不符合一般中文語法
- 在商品名稱中，形容詞的名稱也具有預測能力，例如：咖啡色就常用來形容家具

英文資料篩選

分析前基本處理:

1. 將括號及括號中的字去除
2. 去除標點符號

英文分類模型方法:

方法1 ： 僅做基本處理
- 由於翻譯後幾乎沒and, the等字，因此不需進行初步篩選

方法2 ： 翻譯後只取最後一個字
- 英文語法中，最後一字通常為關鍵字

分析前處理

中文版去除英文，英文版去除中文

```
def  is_chinese(uchar):
     """判断一个unicode是否是汉字"""
     if  uchar  >=  u'\u4e00'  and  uchar  <=  u'\u9fa5':
          return  True
     else:
          return  False

def  is_number(uchar):
     """判断一个unicode是否是数字"""
     if  uchar  >=  u'\u0030'  and  uchar  <=  u'\u0039':
          return  True
     else:
          return  False

def  is_alphabet(uchar):
     """判断一个unicode是否是英文字母"""
     if  (uchar  >=  u'\u0041'  and  uchar  <=  u'\u005a')  or  (uchar  >=  u'\u0061'  and  uchar  <=  u'\u007a'):
          return  True
     else:
          return  False
def  format_str(content):      #only  left  chinese  word
     #content  =  unicode(content,'utf-8')
     content_str  =  ''
     for  i  in  content:
          if  is_chinese(i)  or  is_alphabet(i):
               content_str  =  content_str+i
     return  content_str

def  format_str_ch(content):      #only  left  chinese  word
     #content  =  unicode(content,'utf-8')
     content_str  =  ''
     for  i  in  content:
          if  is_chinese(i):
               content_str  =  content_str+i
     return  content_str

def  format_str_ch_spt(content):      #only  left  chinese  word
     #content  =  unicode(content,'utf-8')
     content_str  =  ''
     for  i  in  content:
          if  is_chinese(i)  or  i  ==  "/":
               content_str  =  content_str+i
     return  content_str
```

主要原因:

- 在貝氏統計模型中，若出現類別有少數英文字，則模型之後則會把有英文字的名詞全部放入該類別，造成預測失效。

次要原因:

- 可減少訓練模型時間

資料切割

```
from sklearn.utils import shuffle

def split_data(data_, num):   #imput whole dataset and percentage of training data
    data_ = shuffle(data_)   #shuffle the data before split
    df1 = data_.iloc[:int(len(data_)*num)]
    df2 = data_.iloc[int(len(data_)*num):]
    df1 = df1.reset_index(drop = True)
    df2 = df2.reset_index(drop = True)
    return df1, df2
```

- 每次切割前會進行洗牌
- 僅依照整體比例切，並不會按照各類別來切
- Kolmogorov–Smirnov Test得知testing set樣本數夠多

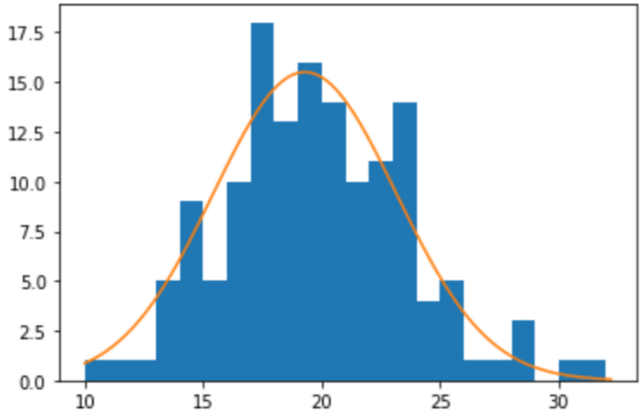
Kolmogorov–Smirnov Test (95% 信心水準)

- 區間錯誤數量：每100筆資料為一單位，而該單位的數值即為該範圍預測錯誤的數量。(隨機變數)
- Kolmogorov–Smirnov Test ：檢驗隨機變數是否符合常態分配
- 中央極限定理 ：若樣本母體不是常態分布，則抽樣數需足夠，抽樣分配才會服從常態分布。
- 抽樣的抽樣皆為常態分布，因此樣本數足夠

Kolmogorov–Smirnov Test (95% 信心水準)

```
import matplotlib.pyplot as plt
plt.hist( partial_wrong, bins=np.arange(np.min(partial_wrong), np.max(partial_wrong)+0.2))
x = np.arange(np.min(partial_wrong), np.max(partial_wrong)+0.2, 0.2)
plt.plot(x, 150*n.pdf(x))
```

```
[<matplotlib.lines.Line2D at 0x7fa29486c9b0>]
```



藍色圖表為資料分布
橘色圖表為常態分布

```
from scipy.stats import kstest
if kstest(partial_wrong, n.cdf).pvalue < 0.05:
    print("is not normal distribution")
else:
    print("is normal distribution")

```

```
is normal distribution
```

樣本數足夠

計分系統

Get the Largest Number

```
rem = []
def get_largest_number(list_,num):
    x = Counter(list_).most_common()
    #print((x))
    try:
        if x[num][1] == x[num+1][1]:
            rem.append("have_the_same")
            #print(num)
            return get_largest_number(list_, num+1)
        else:
            #rem = []
            return x[num][0]
    except:
        try:
            return x[num][0]
        except:
            return "無法分類"
```

演算法說明:

Step1: 找 list 中出現最多次的「名稱」

Step2: 若有多個「名稱」出現次數相同，則在只考慮多個名稱的情況下，取在 list 最後面的「名稱」。

設計假設及原因:

Step1假設: 每個單詞具有相同比重，沒有哪些詞彙特別重要

Step2假設: 越後面的詞彙越重要

New Score Algorithm/ Calculate Word TF-IDF

```
rem_new = []
multi_vote = 0
total_multi = []
def calculate_word_TF_idf(words):
    A = [0]*(Tf_idf.shape[0])
    #print(A)
    for i in range(len(words)):
        col = vectorizer.vocabulary_.get(words[i])
        if col == None:
            continue
        for j in range(Tf_idf.shape[0]):
            A[j] = A[j] + Tf_idf[j][col]
    #print(A)
    return A


def new_score_alg(words, list_, num):    #word: is the original word sent into the model
    x = Counter(list_).most_common()    #list is the solution get by the model
    #print(x)
    try:
          if x[0][1] == x[1][1]:
            rem_new.append("have_the_same")
            get = calculate_word_TF_idf(words)
            return index_list[get.index(max(get))]
          else:
            #multi_vote = multi_vote + 1
            print("multivote")
            total_multi.append("a")
            return x[0][0]
    except:
        print("except")
        #print(calculate_word_TF_idf(words).shape())
        try:
            return x[0][0]
        except:
            return "無法分類"
```

New Score Algorithm 演算法說明:

- Step1: 找出 "list" 中出現最多次的「名稱」
- Step2: 若最多次的「名詞」超過一個，則利用Calculate Word TF-IDF演算法

Calculate Word TF-IDF 演算法說明:

- Step1: 將預測商品切割後的詞彙對應到的各類別TF-IDF值進行加總
- Step2: 取TF-IDF最大值的對應類別的作為商品類別

愛買、大潤發資料

資料分類(16 個類別)

- 3C電子/配件
- 傢俱寢飾
- 大型家電/視聽影音
- 嬰幼兒與孕婦
- 寵物
- 小家電專區
- 居家清潔
- 廚房用品
- 戶外休閒
- 服飾與配件精品
- 泡麵罐頭
- 生活百貨
- 美容沐浴
- 零食點心
- 食品雜貨
- 飲料沖泡

使用模型

模型: 貝氏分群模型

使用原因

- 在自然語言處理時，貝氏分群模型會具有相對良好的效果
- 所需資料少
- 訓練速度較快

限制

- 無法計算每個字的機率、分數，因此當商品同時被分類到很多類別時，則無法進行分類。

中文純貝氏分群模型

```python
from collections import Counter

count = 0
count_rem_w = 0
count_rem = 0
count_rem_1 = 0
count_rem_n = 0
for i in range(len(test)):
    word = str(test.loc[i,"ch_x"])
    #print(word)
    words = word.split("/")
    words = delete_unnecessary_word(words)
    words = [x for x in words if str(x) != 'nan']
    guess = []
    for j in range(len(words)):
        #print(words[j])
        guess.append(classifier_ch.classify(word_feats(words[j])))
    #print(guess, "ans =" ,test.loc[i,"y"], test.loc[i,"x"])
    ans = get_largest_number(guess, 0)
    if len(rem):
        count_rem = count_rem + 1

    if test.loc[i,"y"]==ans:
        count = count + 1
    else:
        if len(rem):
            count_rem_w = count_rem_w + 1
            if len(rem) == 1:
                count_rem_1 = count_rem_1 + 1
            else:
                count_rem_n = count_rem_n + 1
        print("guess = ",ans," ","correct = ",test.loc[i,"y"])
        wrong_pair(test.loc[i,"y"] ,ans)
        print(words)
    rem = []
print("多元判斷的機率 = ", count_rem / len(test))
print("多數決情況成功機率",(count - count_rem + count_rem_w) / (len(test) - count_rem))
print("在錯誤情況下多元判斷失敗機率 = ", count_rem_w / (len(test) - count))
print("多元判斷錯誤下一, 為一元之機率 = ", count_rem_1 / count_rem_w)
print(count/len(test))
```

```
多元判斷的機率 =  0.16322432587492827
多數決情況成功機率 0.8580733630442235
在錯誤情況下多元判斷失敗機率 =  0.4144271570014144
多元判斷錯誤下一，為一元之機率 =  0.6416382252559727
0.7971887550200804
```

預測率

多元判斷 ： 一個商品同時被判為**兩個以上類別**

多數決 ： 一個商品只被判斷到一個類別

一元 ： 一個商品同時被**判為兩個類別**

使用模型

英文純貝氏模型

```
from collections import Counter
count = 0
for i in range(len(test)):
    word = str(test.loc[i,"x"])
    #print(word)
    words = word.split(" ")
    guess = []
    for j in range(len(words)):
        #print(words[j])
        guess.append(classifier.classify(word_feats(words[j])))
    #print(guess, "ans =" ,test.loc[i,"y"], test.loc[i,"x"])
    ans = get_largest_number(guess, 0)
    if test.loc[i,"y"]==ans:
        count = count + 1
    else:
        wrong_pair(test.loc[i,"y"] ,ans)
        print("guess = ",ans," ","correct = ",test.loc[i,"y"])
        print(words)
print(count/len(test))
```

預測率：
0.5140562248995983

使用模型

中文與英文模型比較

```python
count_diff = 0
count_ch = 0
count_eg = 0
count_egR = 0
count_chR = 0
count_RR = 0
count_ww = 0
for i in range(len(test)):
    print(" ")
    word = str(test.loc[i,"ch_x"])
    print(word)
    words = word.split("/")
    ch_guess = []
    for j in range(len(words)):
        #print(words[j])
        ch_guess.append(classifier_ch.classify(word_feats(words[j])))
    #print(guess, "ans =" ,test.loc[i,"y"], test.loc[i,"x"])
    ch_ans = get_largest_number(ch_guess, 0)
    if ch_ans == test.loc[i,"y"]:
        count_what_right_ch(ch_ans)
        count_ch = count_ch + 1

    word = str(test.loc[i,"x"])
    print(word)
    words = word.split(" ")
    guess = []
    for j in range(len(words)):
        #print(words[j])
        guess.append(classifier.classify(word_feats(words[j])))
    #print(guess, "ans =" ,test.loc[i,"y"], test.loc[i,"x"])
    ans = get_largest_number(guess, 0)
    if ans == test.loc[i,"y"]:
        count_what_right_eg(ans)
        count_eg = count_eg + 1

    if ch_ans != ans:
        count_diff = count_diff + 1
        print("chinese_guess", ch_ans)
        print("guess", ans)

    if ch_ans != test.loc[i,"y"] and ans == test.loc[i,"y"]:
        count_egR = count_egR + 1
    if ch_ans == test.loc[i,"y"] and ans != test.loc[i,"y"]:
        count_chR = count_chR + 1
    if ch_ans != test.loc[i,"y"] and ans != test.loc[i,"y"]:
        count_ww = count_ww + 1
    if ch_ans == test.loc[i,"y"] and ans == test.loc[i,"y"]:
        count_RR = count_RR + 1

print("chinese = ",count_ch / len(test))
print("eg = ",count_eg / len(test))
print("only english right", count_egR / len(test))
print("only chinese right", count_chR / len(test))
print("both right", count_RR / len(test))
print("both wrong", count_ww / len(test))
print(count_diff / len(test))
```

```
only english right 0.05794606999426276
only chinese right 0.2719449225473322
both right 0.4561101549053563
both wrong 0.2139988525530694
0.4529546758462421
```

中文模型與英文模型判斷不同的比例

使用模型

中文與英文模型類別預測強弱比較

```
eg Counter({'居家清潔': 471, '美容沐浴': 307, '零食點心': 243, '食品雜貨': 161, '大型家電/視聽影音': 115, '飲料沖泡': 102, '廚房用品': 100, '生活百貨': 89,
ch Counter({'美容沐浴': 510, '居家清潔': 441, '零食點心': 354, '食品雜貨': 264, '生活百貨': 176, '廚房用品': 160, '飲料沖泡': 145, '大型家電/視聽影音': 142,
'傢俱寢飾': 71, '戶外休閒': 43, '服飾與配件精品': 37, '泡麵罐頭': 16, '寵物': 16, '小家電專區': 16, '3C電子/配件': 4, '嬰幼兒與孕婦': 1})
'傢俱寢飾': 88, '戶外休閒': 76, '服飾與配件精品': 57, '泡麵罐頭': 49, '小家電專區': 37, '寵物': 17, '嬰幼兒與孕婦': 12, '3C電子/配件': 10})
```

- 在英文模型中，「居家清潔」的預測率較中文模型高出不少
- 中文模型在資料極度不平衡的情況下分析效果佳

模型: TF-IDF 計分系統

使用原因

- 在分類少數類別時具有良好效果
- 計算時間非常短
- 有實質分數，解決貝氏分群模型「無給分系統」的問題

限制

- 在很多類別的情況下，分類效果不好
- 資料不足時，很多單字會被忽略

中文貝氏與TF-IDF混合模型結果

```
from collections import Counter

count = 0
count_rem_w = 0
count_rem = 0
count_rem_1 = 0
count_rem_n = 0
for i in range(len(test)):
    word = str(test.loc[i,"ch_x"])
    #print(word)
    words = word.split("/")
    words = delete_unnecessary_word(words)
    words = [x for x in words if str(x) != 'nan']
    guess = []
    for j in range(len(words)):
        #print(words[j])
        guess.append(classifier_ch.classify(word_feats(words[j])))
    #print(guess, "ans =" ,test.loc[i,"y"], test.loc[i,"x"])
    ans = new_score_alg(words, guess, 0)
    if len(rem_new):
        count_rem = count_rem + 1

    if test.loc[i,"y"]==ans:
        count = count + 1
    else:
        if len(rem_new):
            count_rem_w = count_rem_w + 1
            if len(rem_new) == 1:
                count_rem_1 = count_rem_1 + 1
            else:
                count_rem_n = count_rem_n + 1
        print("guess = ",ans," ","correct = ",test.loc[i,"y"])
        wrong_pair(test.loc[i,"y"] ,ans)
        print(words)
    rem_new = []
print("多元判斷的機率 = ", count_rem / len(test))
print("多數決情況成功機率",(count - count_rem + count_rem_w) / (len(test) - count_rem))
print("在錯誤情況下多元判斷失敗機率 = ", count_rem_w / (len(test) - count))
print("多元判斷錯誤下一, 為一元之機率 = ", count_rem_1 / count_rem_w)
print(count/len(test))
```

```
多元判斷的機率 =  0.1629374641422834
多數決情況成功機率 0.8577793008910213
在錯誤情況下多元判斷失敗機率 =  0.4503311258278146
多元判斷錯誤下一，為一元之機率 =  1.0
0.7834193918531268
```

預測率

使用模型

中文貝氏與TF-IDF混合模型優劣分析

Final Decision

- 預測率尚有改善空間
- 運算速度佳

使用模型

中英文混和模型結果

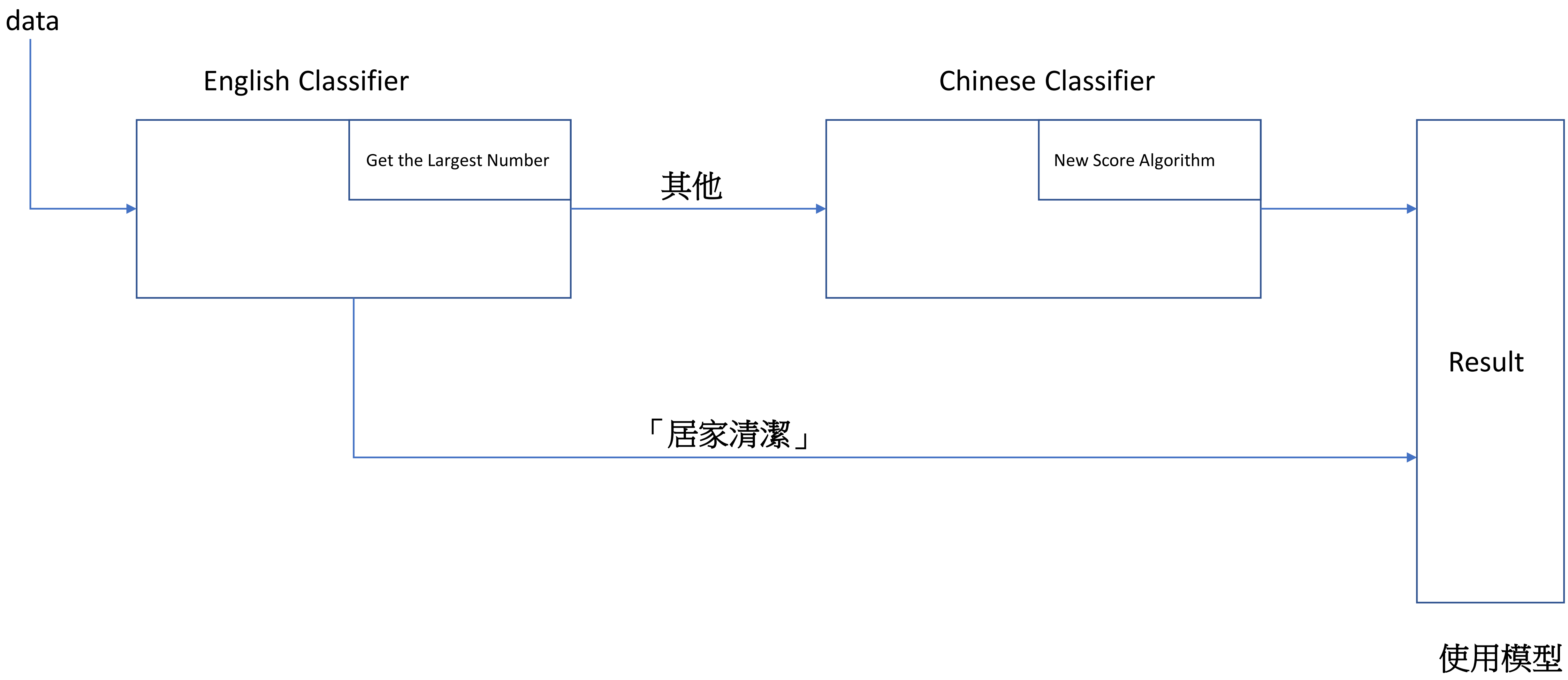
```python
from collections import Counter

count = 0
count_rem_w = 0
count_rem = 0
count_rem_l = 0
count_rem_n = 0
for i in range(len(test)):
    #English modeling
    word = str(test.loc[i,"x"])
    print(word)
    words = word.split(" ")
    guess = []
    for j in range(len(words)):
        #print(words[j])
        guess.append(classifier.classify(word_feats(words[j])))
    #print(guess, "ans =" ,test.loc[i,"y"], test.loc[i,"x"])
    ans = get_largest_number(guess, 0)
    if ans == "居家清潔":
        count = count + 1
        continue

    #Chinese modeling
    word = str(test.loc[i,"ch_x"])
    #print(word)
    words = word.split("/")
    words = delete_unnecessary_word(words)
    words = [x for x in words if str(x) != 'nan']
    guess = []
    for j in range(len(words)):
        #print(words[j])
        guess.append(classifier_ch.classify(word_feats(words[j])))
    #print(guess, "ans =" ,test.loc[i,"y"], test.loc[i,"x"])
    ans = new_score_alg(words, guess, 0)
    if len(rem_new):
        count_rem = count_rem + 1

    if test.loc[i,"y"]==ans:
        count = count + 1
    else:
        if len(rem_new):
            count_rem_w = count_rem_w + 1
            if len(rem_new) == 1:
                count_rem_l = count_rem_l + 1
            else:
                count_rem_n = count_rem_n + 1
        print("guess = ",ans," ","correct = ",test.loc[i,"y"])
        wrong_pair(test.loc[i,"y"] ,ans)
        print(words)
    rem_new = []
print("多元判斷的機率 = ", count_rem / len(test))
print("多數決情況成功機率",(count - count_rem + count_rem_w) / (len(test) - count_rem))
print("在錯誤情況下多元判斷失敗機率 = ", count_rem_w / (len(test) - count))
print("多元判斷錯誤下一, 為一元之機率 = ", count_rem_l / count_rem_w)
print(count/len(test))
```

```
多元判斷的機率 =  0.06855995410212277
多數決情況成功機率 0.9291653834308593
在錯誤情況下多元判斷失敗機率 =  0.3215339233038348
多元判斷錯誤下一，為一元之機率 =  1.0
0.9027538726333907
```

預測率

使用模型

中英文混和模型設計原理



中英文混和模型優劣分析

- 分析預測率高達 90%
- 運算速度十分緩慢，需要花一般模型的兩倍以上的時間，且資料量大時google翻譯過慢

愛買、大潤發、統一超、
部分家樂福資料

資料分類(28 個類別)

- 3C用品
- 乾貨零食
- 化妝用品
- 嬰兒用品
- 家具
- 家電
- 寢具
- 居家清潔
- 居家用品
- 廁所家電
- 廚房煮菜/吃飯用具
- 懶人食物
- 戶外用品
- 收納
- 文具
- 曬衣/洗衣用品
- 浴室/私人清潔
- 男女衣著
- 禮盒
- 肉品海鮮
- 藥品
- 調味料/米
- 配件
- 零食/餅乾
- 食材
- 飲料
- 麵包
- 麵類/煮

使用模型

中文貝氏與TF-IDF混合模型結果(部分分群)

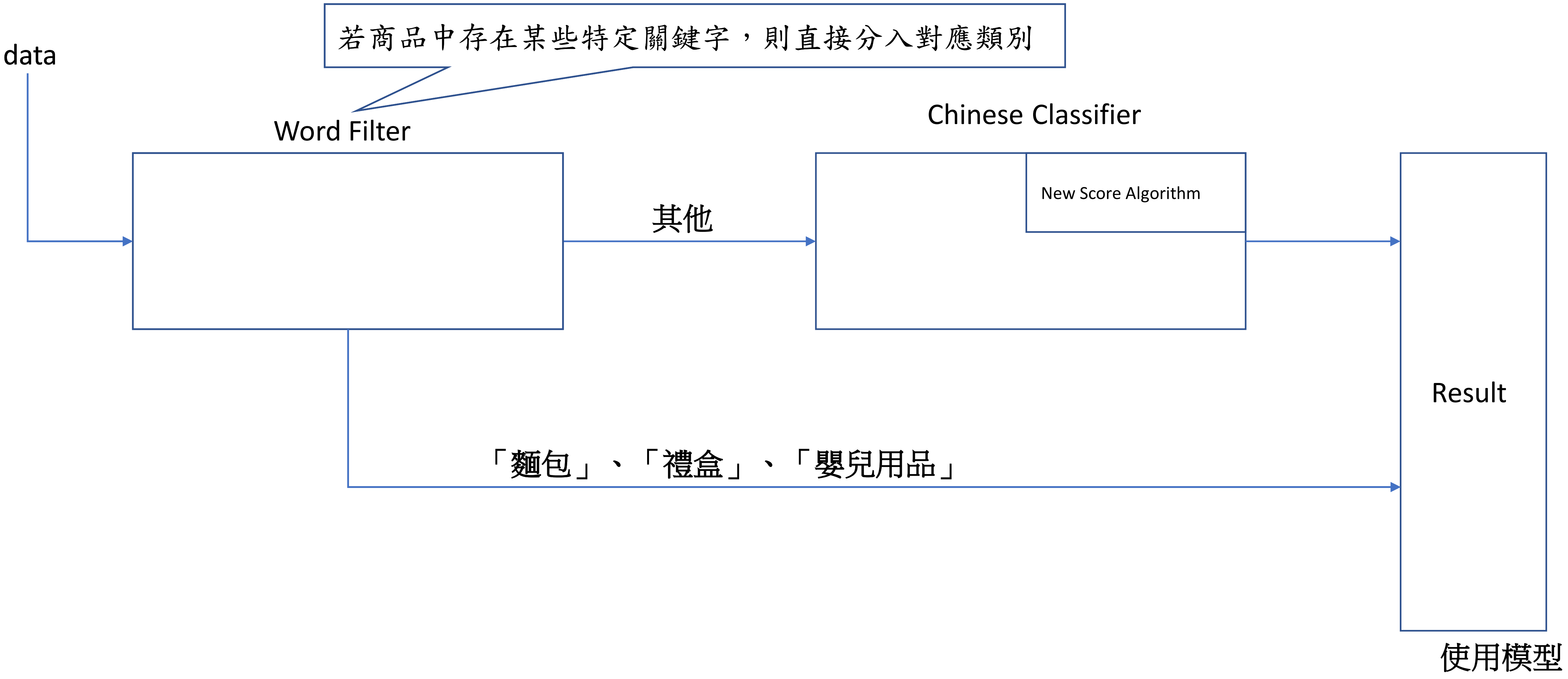
```python

from collections import Counter

count = 0
count_rem_w = 0
count_rem = 0
count_rem_l = 0
count_rem_n = 0
for i in range(len(test)):
    if i % 100 ==0:
        print(i/len(test))
    word = str(test.loc[i,"spt_data"])
    #print(word)
    words = word.split("/")
    words = delete_unnecessary_word(words)
    words = [x for x in words if str(x) != 'nan']
    guess = []
    for j in range(len(words)):
        #print(words[j])
        if words[j] == "麵包":
            guess = ["麵包"]
            break
        elif words[j] == "禮盒":
            guess = ["禮盒"]
            break
        elif words[j] == "童" or words[j] == "女童":
            guess = ["嬰兒用品"]
            break
        guess.append(classifier.classify(word_feats(words[j])))
    #print(guess, "ans =" ,test.loc[i,"y"], test.loc[i,"x"])
    ans = new_score_alg(words, guess, 0)
    if len(rem_new):
        count_rem = count_rem + 1

    if test.loc[i,"cate"]==ans:
        count = count + 1
    else:
        if len(rem_new):
            count_rem_w = count_rem_w + 1
            if len(rem_new) == 1:
                count_rem_l = count_rem_l + 1
            else:
                count_rem_n = count_rem_n + 1
        print("guess = ",ans," ","correct = ",test.loc[i,"cate"])
        wrong_pair(test.loc[i,"cate"] ,ans)
        print(words)
    rem_new = []
print("多元判斷的機率 = ", count_rem / len(test))
print("多數決情況成功機率",(count - count_rem + count_rem_w) / (len(test) - count_rem))
print("在錯誤情況下多元判斷失敗機率 = ", count_rem_w / (len(test) - count))
print("多元判斷錯誤下一, 為一元之機率 = ", count_rem_l / count_rem_w)
print(count/len(test))
```

```
多元判斷的機率 =  0.12067178104913955
多數決情況成功機率 0.8770730173701171
在錯誤情況下多元判斷失敗機率 =  0.3611111111111111
多元判斷錯誤下一，為一元之機率 =  1.0
0.8308106987352271
```

預測率

使用模型

中文貝氏與TF-IDF混合模型原理



程式碼連結(不含資料，僅程式碼)

- https://colab.research.google.com/drive/1nXLX_oGfeCoRaCPfF2_NJWEP_BJJBqSL?usp=sharing（僅含愛買、大潤發版之分群模型）

- https://colab.research.google.com/drive/1nhi_oNihU53I7DgaYtmJo2ECii_a4etx?usp=sharing（含全部超商之分群模型）