In [1]:

```python
# importing block
from pa2 import Dictionary
import requests
import pandas as pd
import numpy as np
from copy import deepcopy
import math
```

In [2]:

```python
def get_training_df(url):
    r = requests.get(url)
    temptext = r.text
    with open("temp.txt", "w") as text_file:
        text_file.write(str(temptext))
    text_file.close()

    templist = temptext.split("\n")
    for i in range(len(templist)):
        templist[i] = templist[i].split()
        if "\r" in templist[i]:
            templist[i].pop[-1]

    training_dict = {}
    for item in templist:
        training_dict[item[0]] = item[1:]

    training_df = pd.DataFrame(training_dict)
    for cat in range(training_df.shape[1]):
        for item in range(training_df.shape[0]):
            training_df.iloc[item, cat] = int(training_df.iloc[item, cat]) -1
    print("training_df created")
    return training_df

def train_term_extract(all_doc, training_df):
    list_list_preprocess = [] ## index 0 means cat1
    for cat in range(training_df.shape[1]):
        list_list_preprocess.append([])
        for index in training_df[str(cat +1)]:
            list_list_preprocess[cat].append(deepcopy(all_doc.preprocess_list[index].word_dic))
            # use .word_dic to get each word
    print("train_term created")
    return list_list_preprocess


def term_cat_making(list_list_dict, cat):
    dict_matrix = {} # key: each term, value: matrix for present or not in each category
    for c in range(cat):
        per_cat_sum = len(list_list_dict[c])
        for doc in range(per_cat_sum):
            for term in list_list_dict[c][doc].keys():
                if term not in dict_matrix:
                    n11 = n10 = n01 = n00 = likelihood_ratio = chi_sqr = np.nan
                    dict_matrix[term] = pd.DataFrame(np.array([[0, per_cat_sum, n11, n10, n01, n00, likelihood_ratio, chi_sqr]]
* cat), columns=["present", "absent", "n11", "n10", "n01", "n00", "likelihood_ratio", "chi_sqr"])
                    index_name = {}
                    for i in range(cat):
                        index_name[i] = "cat_" + str(i)
                    dict_matrix[term] = dict_matrix[term].rename(index=index_name)
                    # print(dict_matrix[term])

                dict_matrix[term]["present"][c] += 1  # df +1
                dict_matrix[term]["absent"][c] -= 1
    print("term_cat_matrix created")
    return dict_matrix

def fill_nx(matrix, cat):
    # get matrix for each term
    # return modified matrix
    for c in range(cat):
        matrix["n11"][c] = matrix["present"][c]
        matrix["n10"][c] = matrix["absent"][c]
        matrix["n01"][c] = matrix["present"].sum() - matrix["n11"][c]
        matrix["n00"][c] = matrix["absent"].sum() - matrix["n10"][c]

    return matrix
```

In [3]:

```python
def cal_ratio(matrix, cat):
    # get matrix for each term
    # return likelihood ratio of term in all cat
    if matrix["n00"].sum() == 0:
        matrix = fill_nx(matrix, cat)

    for cat in range(matrix.shape[0]):
        n11 = matrix["n11"][cat]
        n10 = matrix["n10"][cat]
        n01 = matrix["n01"][cat]
        n00 = matrix["n00"][cat]
        N = n11 + n10 + n01 + n00

        ## likelihood ratio
        pt = (n11 + n01) / N
        p1 = n11 / (n11 + n10)
        p2 = n01 / (n01 + n00)

        likelihood = -2 * np.log10((((pt ** n11) * (1 - pt) ** n10) * ((pt ** n01) * (1 - pt) ** n00)) /
                                  (((p1**n11) * (1 - p1)**n10) * ((p2**n01) * (1 - p2)**n00)))
        matrix["likelihood_ratio"][cat] = likelihood

        ## chi square
        e11 = (n11 + n01) * (n11 + n10) / N
        e10 = (n11 + n10) * (n00 + n10) / N
        e01 = (n11 + n01) * (n01 + n00) / N
        e00 = (n00 + n01) * (n00 + n10) / N

        chi_square = (n11 - e11)**2 / e11 + (n10 - e10)**2 / e11 + (n01 - e01)**2 / e11 + (n00 - e00)**2 / e11
        matrix["chi_sqr"][cat] = chi_square

    return matrix
```

In [4]:

```python
def test_term_extract(all_doc, training_df):
    temp = []
    for index, row in training_df.iterrows():
        temp.append(row.to_list())
    train_ids = []
    for row in temp:
        for item in row:
            train_ids.append(item)

    dict_dict = {} ## key:doc_id(in real), values:preprocess_list
    for index in range(len(all_doc.preprocess_list)):
        if index in train_ids:
            continue
        dict_dict[index +1] = all_doc.preprocess_list[index].word_dic
        # use .word_dic to get each word
    return dict_dict
```

In [5]:

```python
def feature_selection(dict_df, method="sum_likelihood"):
    store_score = {}
    feature_list = []
    feature_upper = 500

    if method == "sum_likelihood":
        ## top 500 from sum of the likelihood ratio
        for term in dict_df.keys():
            store_score[term] = dict_df[term]["likelihood_ratio"].sum()
        feature_list = sorted(store_score.items(), key=lambda item: item[1], reverse=True)
        for rank in range(feature_upper):
            feature_list.append(feature_list[rank][0])

    if method == "max_likelihood":
        ## top 500 from the likelihood ratio
        for term in dict_df.keys():
            store_score[term] = dict_df[term]["likelihood_ratio"].max()
        feature_list = sorted(store_score.items(), key=lambda item: item[1], reverse=True)
        for rank in range(feature_upper):
            feature_list.append(feature_list[rank][0])

    if method == "max_chi":
        ## top 500 from the chi_sqr ratio
        for term in dict_df.keys():
            store_score[term] = dict_df[term]["chi_sqr"].max()
        feature_list = sorted(store_score.items(), key=lambda item: item[1], reverse=True)
        for rank in range(feature_upper):
            feature_list.append(feature_list[rank][0])

    if method == "hy_max":
        ## top 500 from the hybrid ratio
        store_score_like = {}
        store_score_chi = {}
        for term in dict_df.keys():
            store_score_like[term] = dict_df[term]["likelihood_ratio"].max()
            store_score_chi[term] = dict_df[term]["chi_sqr"].max()
        rank_like = sorted(store_score_like.items(), key=lambda item: item[1], reverse=True)
        rank_chi = sorted(store_score_chi.items(), key=lambda item: item[1], reverse=True)
        buffer = set()
        def hybrid_rank(check, rank_list, term):
            if term in check:
                check.discard(term)
                rank_list.append(term)
            else:
                check.add(term)

        for i in range(len(rank_chi)):
            hybrid_rank(buffer, feature_list, rank_chi[i][0])
            hybrid_rank(buffer, feature_list, rank_like[i][0])
            if len(feature_list) == feature_upper:
                break

    print("feature_selection with %s completed" %method)
    return feature_list
```

In [6]:

```python
def cal_condprob(cat, list_list_preprocess, list_term):
    tf_dict_list = {}
    dict_list = {}
    for term in list_term:
        tf_dict_list[term] = [0] * cat
        dict_list[term] = [0] * cat
    for c in range(cat):
        for doc in list_list_preprocess[c]:
            for term in doc.keys():
                if term in list_term:
                    tf = len(doc[term])
                    tf_dict_list[term][c] += tf
        for term in list_term:
            dict_list[term][c] = (int(tf_dict_list[term][c]) + 1) / (sum(tf_dict_list[term][0: -1]) + len(list_term))
    print("cal_condprob completed")
    return dict_list
```

In [7]:

```python
def apply_multi_NB(cat, features, condprob, test_doc):
    extracted = []
    score = [0] * cat
    for term in test_doc:
        if term in features:
            extracted.append(term)
    for c in range(cat):
        for term in extracted:
            score[c] += math.log10(condprob[term][c])
    predict = 0
    for c in range(cat):
        if score[c] >= score[predict]:
            predict = c
    return predict+1 ## right class for the result
```

In [9]:

```python
# Press the green button in the gutter to run the script.
if __name__ == '__main__':

    url = "https://ceiba.ntu.edu.tw/course/88ca22/content/training.txt"
    training_df = get_training_df(url)

    all_doc = Dictionary()
    all_doc.preprocess_all_file("D:\Desktop\IR\PA2\IRTM")

    train_term = train_term_extract(all_doc, training_df) ## type: list(cat) of list(docs) of dict(word_dic)
    cat = len(train_term)
    term_cat_matrix = term_cat_making(train_term, cat)  ## type: dict, make a matrix for each term


    for term in term_cat_matrix.keys():
        ## traverse all terms to calculate likelihood
        term_cat_matrix[term] = cal_ratio(term_cat_matrix[term], cat)
    print("cal_likelihood completed")

    ## feature selection
    features = feature_selection(term_cat_matrix, method="max_chi") ## type: list

    prior = 1/13
    ## prior of each category is identical, so we skip prior

    condprob = cal_condprob(cat, train_term, features) ## type: dict(term) of list(probability for each cat)

    #################################
    ########### TEST PART ###########
    #################################
    test_term = test_term_extract(all_doc, training_df) ## tpye: dict, key: real doc name/ id, value: word_dict

    #################################
    ######## MAKE PREDICTION ########
    #################################

    predicts = []
    id = []
    for key in test_term.keys():
        id.append(key)
        predicts.append(apply_multi_NB(cat, features, condprob, test_term[key].keys()))
    result = pd.DataFrame()
    result["Id"] = id
    result["Value"] = predicts
    print("prediction made")

    file_name = "result_05_max_chi.csv"
    result.to_csv(file_name, index=False)
    print(file_name, "file saved")
```

```
training_df created
train_term created
term_cat_matrix created
cal_likelihood completed
feature_selection with max_chi completed
cal_condprob completed
prediction made
result_05_max_chi.csv file saved
```

用字典把每個訓練用的 term 存你來，並用表的方式計算 likelihood ratio, chi_square

In [11]:

```python
term_cat_matrix["navi"]
```

Out[11]:

	present	absent	n11	n10	n01	n00	likelihood_ratio	chi_sqr
cat_0	7.0	8.0	7.0	8.0	20.0	160.0	4.575504	46.678063
cat_1	0.0	15.0	0.0	15.0	27.0	153.0	2.027056	8.307692
cat_2	0.0	15.0	0.0	15.0	27.0	153.0	2.027056	8.307692
cat_3	3.0	12.0	3.0	12.0	24.0	156.0	0.203189	1.641026
cat_4	0.0	15.0	0.0	15.0	27.0	153.0	2.027056	8.307692
cat_5	0.0	15.0	0.0	15.0	27.0	153.0	2.027056	8.307692
cat_6	0.0	15.0	0.0	15.0	27.0	153.0	2.027056	8.307692
cat_7	5.0	10.0	5.0	10.0	22.0	158.0	1.767034	16.455840
cat_8	12.0	3.0	12.0	3.0	15.0	165.0	16.750614	189.641026
cat_9	0.0	15.0	0.0	15.0	27.0	153.0	2.027056	8.307692
cat_10	0.0	15.0	0.0	15.0	27.0	153.0	2.027056	8.307692
cat_11	0.0	15.0	0.0	15.0	27.0	153.0	2.027056	8.307692
cat_12	0.0	15.0	0.0	15.0	27.0	153.0	2.027056	8.307692