Teerameth Rassameecharoenchai

61340500032

2 Sentiment Analysis

2.2 Movie Review Data

Let us first start by looking at the data provided with the exercise. We have positive and negative movie reviews labeled by human readers, all positive and negative reviews are in the ‘pos’ and ‘neg’ folders respectively. If you look in- side a sample file, you will see that these review messages have been ‘tokenized’, where all words are separated from punctuations. There are approximately 1000 files in each category with files names starting with cv000, cv001, cv002 and so on. You will split the dataset into training set and testing set.

1. Write some code to load the data from text files.

1

```
import glob, os

import sklearn.model_selection
from matplotlib import pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, log_loss
from sklearn.model_selection import cross_val_predict, cross_validate, KFold
import pandas as pd
import numpy as np
folders_path = "./review_polarity/txt_sentoken/"
pos_path = os.path.join(os.getcwd(), (folders_path + 'pos'))
neg_path = os.path.join(os.getcwd(), (folders_path + 'neg'))
pos_files = glob.glob(pos_path + '/*.txt')
neg_files = glob.glob(neg_path + '/*.txt')
data, label = [], []
# วน loop เพื่อ load ไฟล์ (ได้มาเป็น list ของ string ที่แสดงถึงแต่ละบรรทัดในเอกสารแล้วจับรวมกันเป็น string 1 อันต่อเอกสา
for file in pos_files:
    f = open(file,"r")
    lines = f.readlines()
    string = ""
    for line in lines: string += line
    data.append(string)
    label.append(1) # label 1 as positive
for file in neg_files:
    f = open(file,"r")
    lines = f.readlines()
    string = ""
```

```
    for line in lines: string += line
    data.append(string)
    label.append(0) # label 0 as negative

# Train-Test split 80:20
train, test, label_train, label_test = sklearn.model_selection.train_test_split(data, label, test_size=0.
```

2.3 TF-IDF

From a raw text review, you want to create a vector, whose elements indicate the number of each word in each document. The frequency of all words within the documents are the ‘features’ of this machine learning problem.

A popular method for transforming a text to a vector is called tf-idf, short for term frequencyinverse document frequency.

1. Conduct a research about tf-idf and explain how it works.
2. Scikit-learn provides a module for calculating this, this is called TfidfVec- torizer. You can study how this function is used here:

`http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html`

Write code to transform your text to tf-idf vector.

2
```
# สรุป tf-idf ที่ได้อ่านจาก towarddatascience.com เพื่อแปลงเอกสารเป็น tf-idf vector สำหรับนำไปใช้กับ model
# tf(t,d) = count of t in d / number of words in d
# Document Frequency: df(t) = occurrecnce of t in documents
# Inverse Document Frequency: idf(t) = N/df = log(N/(df+1))
# td-idf(t,d) = tf(t,d)*log(N/(df+1))
# Ref: https://towardsdatascience.com/tf-idf-for-document-ranking-from-scratch-in-python-on-real-world-da
from sklearn.feature_extraction.text import TfidfVectorizer , TfidfTransformer
# Pipeline vectorizer object & Compute TF-IDF scores
vectorizer = TfidfVectorizer(analyzer='word' , stop_words='english')
tf_trans = TfidfTransformer()
tf_idf_vector_train = tf_trans.fit_transform(vectorizer.fit_transform(train))
tf_idf_vector_test = tf_trans.fit_transform(vectorizer.transform(test)) # test set ใช้ vectorizer ตัวเดียวกับ
# แปลงเป็น tf-idf vector แล้วก็แปลงเป็น pandas dataframe ต่อ
X_train = pd.DataFrame(tf_idf_vector_train.toarray(), columns=vectorizer.get_feature_names())
X_test = pd.DataFrame(tf_idf_vector_test.toarray(), columns=vectorizer.get_feature_names())
y_train = np.ravel(pd.DataFrame(np.asarray(label_train)))
y_test = np.ravel(pd.DataFrame(np.asarray(label_test)))
[X_train.shape, X_test.shape, y_train.shape, y_test.shape]
```

2
```
[(1600, 35948), (400, 35948), (1600,), (400,)]
```

2.4 Classification

Use 4 different models to classify each movie into positive or negative category.

1. K-Nearestneighbormodel,using module `sklearn.neighbors.KNeighborsClassifier`
2. RandomForest, using module `sklearn.ensemble.RandomForestClassifier`
3. SVM, using module `sklearn.svm.SVC`
4. Neural network, using `sklearn.neural_network.MLPClassifier`

You may pick other models you would like to try. Just present results for at least 4 models. Please provide your code for model fitting and cross validation. Calculate your classification accuracy, precision, and recall.

3

```
# ทำ cross validation แล้วเอามาทดสอบด้วย test set อีกทีโดยทำเป็น classification report เพื่อให้ได้ทั้ง accuracy, p
# Build 4 models and store in list
knc = KNeighborsClassifier(n_neighbors=5)
rfc = RandomForestClassifier(n_estimators=100, random_state=136)
svc = SVC(kernel='rbf', random_state=136)
mlpc = MLPClassifier(hidden_layer_sizes=(64,), activation='relu', early_stopping=True, n_iter_no_change=2
model_list = [knc, rfc, svc, mlpc]
model_names = ['K Neighbors Classifier', 'Random Forest Classifier', 'SVC', 'Neural Network']
for i, model in enumerate(model_list): # Loop all models in list
    # Apply cross validation using K=5
    for j, model in enumerate(cross_validate(model, X_train, y_train, cv=5, return_estimator=True)['estim
        y_pred = model.predict(X_test)
        print("Model name: %s [Fold %d]"%(model_names[i], j))
        print("Confusion Matrix")
        print(confusion_matrix(y_test, y_pred))
        print("Classification report")
        print(classification_report(y_test, y_pred, target_names=['Neg', 'Pos']))
```

```
Model name: K Neighbors Classifier [Fold 0]
Confusion Matrix
[[121  77]
 [ 73 129]]
Classification report
              precision    recall  f1-score   support

         Neg       0.62      0.61      0.62       198
         Pos       0.63      0.64      0.63       202

    accuracy                           0.62       400
   macro avg       0.62      0.62      0.62       400
weighted avg       0.62      0.62      0.62       400

Model name: K Neighbors Classifier [Fold 1]
Confusion Matrix
[[114  84]
 [ 67 135]]
Classification report
              precision    recall  f1-score   support

         Neg       0.63      0.58      0.60       198
         Pos       0.62      0.67      0.64       202

    accuracy                           0.62       400
   macro avg       0.62      0.62      0.62       400
weighted avg       0.62      0.62      0.62       400

Model name: K Neighbors Classifier [Fold 2]
Confusion Matrix
[[120  78]
 [ 75 127]]
```

```
Classification report
              precision    recall  f1-score   support

         Neg       0.62      0.61      0.61       198
         Pos       0.62      0.63      0.62       202

    accuracy                           0.62       400
   macro avg       0.62      0.62      0.62       400
weighted avg       0.62      0.62      0.62       400

Model name: K Neighbors Classifier [Fold 3]
Confusion Matrix
[[110  88]
 [ 80 122]]
Classification report
              precision    recall  f1-score   support

         Neg       0.58      0.56      0.57       198
         Pos       0.58      0.60      0.59       202

    accuracy                           0.58       400
   macro avg       0.58      0.58      0.58       400
weighted avg       0.58      0.58      0.58       400

Model name: K Neighbors Classifier [Fold 4]
Confusion Matrix
[[128  70]
 [ 67 135]]
Classification report
              precision    recall  f1-score   support

         Neg       0.66      0.65      0.65       198
         Pos       0.66      0.67      0.66       202

    accuracy                           0.66       400
   macro avg       0.66      0.66      0.66       400
weighted avg       0.66      0.66      0.66       400

Model name: Random Forest Classifier [Fold 0]
Confusion Matrix
[[175  23]
 [ 77 125]]
Classification report
              precision    recall  f1-score   support

         Neg       0.69      0.88      0.78       198
         Pos       0.84      0.62      0.71       202

    accuracy                           0.75       400
   macro avg       0.77      0.75      0.75       400
weighted avg       0.77      0.75      0.75       400

Model name: Random Forest Classifier [Fold 1]
Confusion Matrix
[[176  22]
 [ 71 131]]
Classification report
              precision    recall  f1-score   support

         Neg       0.71      0.89      0.79       198
         Pos       0.86      0.65      0.74       202

    accuracy                           0.77       400
   macro avg       0.78      0.77      0.76       400
weighted avg       0.79      0.77      0.76       400
```

```
Model name: Random Forest Classifier [Fold 2]
Confusion Matrix
[[177  21]
 [ 69 133]]
Classification report
              precision    recall  f1-score   support

         Neg       0.72      0.89      0.80       198
         Pos       0.86      0.66      0.75       202

    accuracy                           0.78       400
   macro avg       0.79      0.78      0.77       400
weighted avg       0.79      0.78      0.77       400

Model name: Random Forest Classifier [Fold 3]
Confusion Matrix
[[177  21]
 [ 79 123]]
Classification report
              precision    recall  f1-score   support

         Neg       0.69      0.89      0.78       198
         Pos       0.85      0.61      0.71       202

    accuracy                           0.75       400
   macro avg       0.77      0.75      0.75       400
weighted avg       0.77      0.75      0.75       400

Model name: Random Forest Classifier [Fold 4]
Confusion Matrix
[[173  25]
 [ 73 129]]
Classification report
              precision    recall  f1-score   support

         Neg       0.70      0.87      0.78       198
         Pos       0.84      0.64      0.72       202

    accuracy                           0.76       400
   macro avg       0.77      0.76      0.75       400
weighted avg       0.77      0.76      0.75       400

Model name: SVC [Fold 0]
Confusion Matrix
[[160  38]
 [ 51 151]]
Classification report
              precision    recall  f1-score   support

         Neg       0.76      0.81      0.78       198
         Pos       0.80      0.75      0.77       202

    accuracy                           0.78       400
   macro avg       0.78      0.78      0.78       400
weighted avg       0.78      0.78      0.78       400

Model name: SVC [Fold 1]
Confusion Matrix
[[151  47]
 [ 39 163]]
Classification report
              precision    recall  f1-score   support

         Neg       0.79      0.76      0.78       198
```

```
         Pos       0.78      0.81      0.79       202

    accuracy                           0.79       400
   macro avg       0.79      0.78      0.78       400
weighted avg       0.79      0.79      0.78       400

Model name: SVC [Fold 2]
Confusion Matrix
[[161  37]
 [ 46 156]]
Classification report
              precision    recall  f1-score   support

         Neg       0.78      0.81      0.80       198
         Pos       0.81      0.77      0.79       202

    accuracy                           0.79       400
   macro avg       0.79      0.79      0.79       400
weighted avg       0.79      0.79      0.79       400

Model name: SVC [Fold 3]
Confusion Matrix
[[156  42]
 [ 43 159]]
Classification report
              precision    recall  f1-score   support

         Neg       0.78      0.79      0.79       198
         Pos       0.79      0.79      0.79       202

    accuracy                           0.79       400
   macro avg       0.79      0.79      0.79       400
weighted avg       0.79      0.79      0.79       400

Model name: SVC [Fold 4]
Confusion Matrix
[[155  43]
 [ 44 158]]
Classification report
              precision    recall  f1-score   support

         Neg       0.78      0.78      0.78       198
         Pos       0.79      0.78      0.78       202

    accuracy                           0.78       400
   macro avg       0.78      0.78      0.78       400
weighted avg       0.78      0.78      0.78       400

Model name: Neural Network [Fold 0]
Confusion Matrix
[[149  49]
 [ 39 163]]
Classification report
              precision    recall  f1-score   support

         Neg       0.79      0.75      0.77       198
         Pos       0.77      0.81      0.79       202

    accuracy                           0.78       400
   macro avg       0.78      0.78      0.78       400
weighted avg       0.78      0.78      0.78       400

Model name: Neural Network [Fold 1]
Confusion Matrix
[[149  49]
```

```
 [ 35 167]]
Classification report
              precision    recall  f1-score   support

         Neg       0.81      0.75      0.78       198
         Pos       0.77      0.83      0.80       202

    accuracy                           0.79       400
   macro avg       0.79      0.79      0.79       400
weighted avg       0.79      0.79      0.79       400

Model name: Neural Network [Fold 2]
Confusion Matrix
[[149  49]
 [ 45 157]]
Classification report
              precision    recall  f1-score   support

         Neg       0.77      0.75      0.76       198
         Pos       0.76      0.78      0.77       202

    accuracy                           0.77       400
   macro avg       0.77      0.76      0.76       400
weighted avg       0.77      0.77      0.76       400

Model name: Neural Network [Fold 3]
Confusion Matrix
[[130  68]
 [ 24 178]]
Classification report
              precision    recall  f1-score   support

         Neg       0.84      0.66      0.74       198
         Pos       0.72      0.88      0.79       202

    accuracy                           0.77       400
   macro avg       0.78      0.77      0.77       400
weighted avg       0.78      0.77      0.77       400

Model name: Neural Network [Fold 4]
Confusion Matrix
[[144  54]
 [ 36 166]]
Classification report
              precision    recall  f1-score   support

         Neg       0.80      0.73      0.76       198
         Pos       0.75      0.82      0.79       202

    accuracy                           0.78       400
   macro avg       0.78      0.77      0.77       400
weighted avg       0.78      0.78      0.77       400
```

2.5 Model Tuning

Can you try to beat the simple model you created above? Here are some things you may try:

- When creating TfidfVectorizer object, you may tweak sublinear_tf parameter which use the tf with logarithmic scale instead of the usual tf.

- You may also exclude words that are too frequent or too rare, by adjusting max_df and min_df.
- Adjusting parameters available in the model, like neural network structure or number of trees in the forest.

Design at least 3 experiments using these techniques. Show your experimental results.

แยก dataset ใหม่เป็น train-validation-test ในอัตราส่วน 70:20:10 โดยใช้ validation set เป็นตัววัดว่าควรเลือก parameter อันไหนดีโดยใช้ log_loss เป็นเกณฑ์ จากนั้นจึงทดสอบด้วย test set อีกที ตอนแปลง dataset เป็น tf-idf vector ก็ใช้ feture เดียวกันกับ training set ทั้งหมด

4

```
## Use 1+log(tf) by setting sublinear_tf=True
vectorizer = TfidfVectorizer(analyzer='word',
                             stop_words='english',
                             sublinear_tf=True,
                             max_df=0.9, # exclude too frequence words (more than 0.9 by proportion of dc
                             min_df=0.1) # exclude less sequence words (less than 0.1 by proportion of dc
tf_trans = TfidfTransformer()
tf_idf_vector=tf_trans.fit_transform(vectorizer.fit_transform(train))
X = pd.DataFrame(tf_idf_vector.toarray(), columns=vectorizer.get_feature_names())
y = np.ravel(pd.DataFrame(np.asarray(label)))

## Train-Val-Test split 70:20:10
train, buff, label_train, label_buff = sklearn.model_selection.train_test_split(data, label, test_size=0.
val, test, label_val, label_test = sklearn.model_selection.train_test_split(buff, label_buff, test_size=0
vectorizer = TfidfVectorizer(analyzer='word' , stop_words='english')
tf_trans = TfidfTransformer()
tf_idf_vector_train = tf_trans.fit_transform(vectorizer.fit_transform(train))
tf_idf_vector_val = tf_trans.fit_transform(vectorizer.transform(val))
tf_idf_vector_test = tf_trans.fit_transform(vectorizer.transform(test))
X_train = pd.DataFrame(tf_idf_vector_train.toarray(), columns=vectorizer.get_feature_names())
X_val = pd.DataFrame(tf_idf_vector_val.toarray(), columns=vectorizer.get_feature_names())
X_test = pd.DataFrame(tf_idf_vector_test.toarray(), columns=vectorizer.get_feature_names())
y_train = np.ravel(pd.DataFrame(np.asarray(label_train)))
y_val = np.ravel(pd.DataFrame(np.asarray(label_val)))
y_test = np.ravel(pd.DataFrame(np.asarray(label_test)))
[X_train.shape, X_test.shape, y_train.shape, y_test.shape]
```

4

```
[(1400, 34085), (198, 34085), (1400,), (198,)]
```

Tune KNC for number of neighbors วน loop ลองใช้ค่า K ต่าง ๆ แล้วนำมา plot graph ควรจะได้กราฟที่จะเริ่มมี log_loss นิ่งเมื่อ K มากถึงระดับนึง (ถ้า K น้อยจะทำให้ noise มีผลมากแต่ถ้า K มากไปจะทำให้ใช้ทรัพยากรมาก)

5

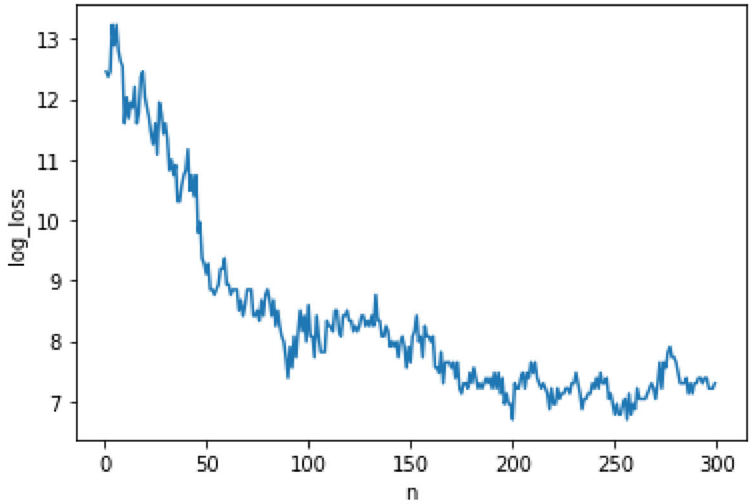
```
log = {'n':[], 'log_loss':[]}
for n in range(1, 300):
    model = KNeighborsClassifier(n_neighbors=n)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_val)
    acc = log_loss(y_val, y_pred)
    print("n=%d: %f"%(n,acc), end='\r')
    log['n'].append(n)
    log['log_loss'].append(acc)
plt.plot(log['n'], log['log_loss'])
```

```
plt.xlabel('n')
plt.ylabel('log_loss')
```

```
n=299: 7.303059
```

5

```
Text(0, 0.5, 'log_loss')
```



เลือก n=100 เนื่องจากมีค่าน้อย(จะได้ใช้ทรัพยากรน้อย)แต่ให้ loss น้อยพอสมควรใน validation set

6

```
model = KNeighborsClassifier(n_neighbors=100)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Confusion Matrix")
print(confusion_matrix(y_test, y_pred))
print("Classification report")
print(classification_report(y_test, y_pred, target_names=['Neg', 'Pos']))
```

```
Confusion Matrix
[[78 20]
 [26 74]]
Classification report
              precision    recall  f1-score   support

         Neg       0.75      0.80      0.77        98
         Pos       0.79      0.74      0.76       100

    accuracy                           0.77       198
   macro avg       0.77      0.77      0.77       198
weighted avg       0.77      0.77      0.77       198
```

Tune Random Forest Classification วน loop ทดลองกำหนดจำนวนต้นไม้แล้ว plot log_loss เทียบด้วยเหตุผลเดียวกับ KNC

7

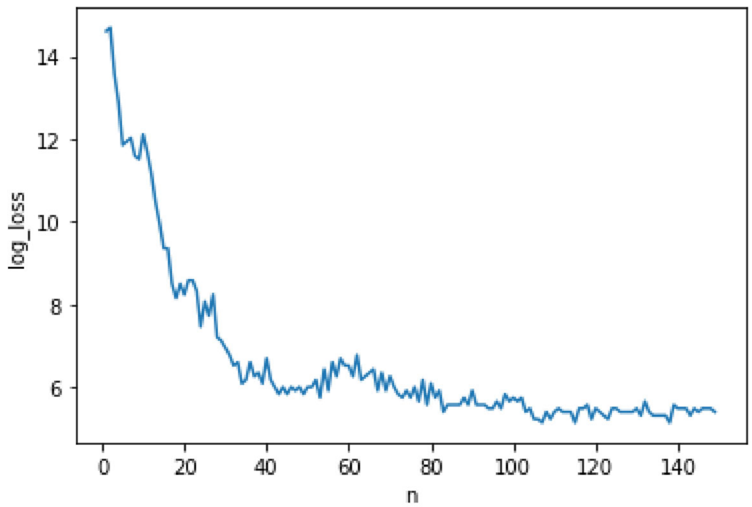
```
log = {'n':[], 'log_loss':[]}
for n in range(1, 150):
    model = RandomForestClassifier(n_estimators=n, random_state=136)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_val)
    acc = log_loss(y_val, y_pred)
    print("n=%d: %f"%(n,acc), end="\r")
    log['n'].append(n)
    log['log_loss'].append(acc)
```

```
plt.plot(log['n'], log['log_loss'])
plt.xlabel('n')
plt.ylabel('log_loss')
```

```
n=149: 5.412843
```

7

```
Text(0, 0.5, 'log_loss')
```



เลือกจำนวน tree = 40 เนื่องจาก log_loss เริ่มลู่เข้าพอดีเมื่อทดสอบกับ validation set (ถ้าใช้จำนวน tree มากเกินไปจะใช้ทรัพยากรมากไปด้วย) และได้ผลลัพทธ์จาก test set ดังนี้

8

```
model = RandomForestClassifier(n_estimators=40, random_state=136)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Confusion Matrix")
print(confusion_matrix(y_test, y_pred))
print("Classification report")
print(classification_report(y_test, y_pred, target_names=['Neg', 'Pos']))
```

```
Confusion Matrix
[[84 14]
 [44 56]]
Classification report
              precision    recall  f1-score   support

         Neg       0.66      0.86      0.74        98
         Pos       0.80      0.56      0.66       100

    accuracy                           0.71       198
   macro avg       0.73      0.71      0.70       198
weighted avg       0.73      0.71      0.70       198
```

Tune SVM for kernel ลองเลือกใช้ kernel แบบต่าง ๆ เพื่อดูว่า fit ด้วยอะไรแล้วเข้ากับข้อมูลได้ดีที่สุด

9

```
kernel_list = ['linear', 'poly', 'rbf', 'sigmoid']
log_loss_list = []
model_list = []
for kernel in kernel_list:
    model = SVC(kernel=kernel, random_state=136)
    model.fit(X_train, y_train)
    model_list.append(model)
    y_pred = model.predict(X_val)
```

```
    acc = log_loss(y_val, y_pred)
    print("kernel=%s: %f"%(kernel,acc))
    log_loss_list.append(acc)
selected_kernel = kernel_list[log_loss_list.index(min(log_loss_list))]
print("Select "+selected_kernel+" as kernel due to lowest log_loss in validation set.")
```

```
kernel=linear: 6.959395
kernel=poly: 11.942608
kernel=rbf: 7.388998
kernel=sigmoid: 7.388990
Select linear as kernel due to lowest log_loss in validation set.
```

10
```
model = model_list[log_loss_list.index(min(log_loss_list))] # pick best model to test
y_pred = model.predict(X_test)
print("Confusion Matrix")
print(confusion_matrix(y_test, y_pred))
print("Classification report")
print(classification_report(y_test, y_pred, target_names=['Neg', 'Pos']))
```

```
Confusion Matrix
[[77 21]
 [19 81]]
Classification report
              precision    recall  f1-score   support

         Neg       0.80      0.79      0.79        98
         Pos       0.79      0.81      0.80       100

    accuracy                           0.80       198
   macro avg       0.80      0.80      0.80       198
weighted avg       0.80      0.80      0.80       198
```

Tune Neural Network hyperparameter using grid search (hidden layer size ,activation function, solver, learning rate) (นานมากกกก)

11
```
# Ref:https://panjeh.medium.com/scikit-learn-hyperparameter-optimization-for-mlpclassifier-4d670413042b
mlp_gs = MLPClassifier(max_iter=1000, early_stopping=True, n_iter_no_change=20, random_state=136) # Setup
parameter_space = {
    'hidden_layer_sizes': [(64,32,64),(64,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.001, 0.005, 0.01, 0.05],
    'learning_rate': ['constant','adaptive'],
}
from sklearn.model_selection import GridSearchCV
clf = GridSearchCV(mlp_gs, parameter_space, n_jobs=-1, cv=5) # 5-folds of K-fold validation
clf.fit(X_train, y_train)
print('Best parameters found:\n', clf.best_params_)
```

```
Best parameters found:
 {'activation': 'relu', 'alpha': 0.005, 'hidden_layer_sizes': (64, 32, 64), 'learning_rate': 'constant',
```

12
```
mlp_gs_best = clf.best_estimator_
y_pred = mlp_gs_best.predict(X_test)
print("Confusion Matrix")
print(confusion_matrix(y_test, y_pred))
print("Classification report")
print(classification_report(y_test, y_pred, target_names=['Neg', 'Pos']))
```

```
Confusion Matrix
[[79 19]
 [21 79]]
Classification report
              precision    recall  f1-score   support

         Neg       0.79      0.81      0.80        98
         Pos       0.81      0.79      0.80       100

    accuracy                           0.80       198
   macro avg       0.80      0.80      0.80       198
weighted avg       0.80      0.80      0.80       198
```

3 Text Clustering

We have heard about Google News clustering. In this exercise, we are going to implement it with Python.

3.1 Data Preprocessing

Let’s switch up and use another dataset called 20newsgroup data, which is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups. The data is collected from a university’s mailing list, where students exchange opinions in everything from motorcycles to middle east politics.

1. Import data using sklearn.datasets.fetch_20newsgroups
2. Transform data to vector with TfidfVectorizer

load dataset มาแล้วทำเหมือนเดิมคือแปลงให้เป็น tf-idf vector แต่คราวนี้ไม่ต้องแบ่ง train-test แล้ว เพราะเรากำลังทำ clustering อยู่

13

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer , TfidfTransformer

newsgroups = fetch_20newsgroups(subset='train')
vectorizer = TfidfVectorizer(analyzer='word',
                             stop_words='english',
                             max_df=0.9, # exclude too frequence words (more than 0.99 by proportion of d
                             min_df=0.1) # exclude less frequence words (less than 0.01 by proportion of
vectors = vectorizer.fit_transform(newsgroups.data)
print(vectors.shape)
```

```
(11314, 62)
```

3.2 Clustering

We are going to use the simplest clustering model, k-means clustering, to do this task. Our hope is that this simple algorithm will result in meaningful news categories, without using labels.

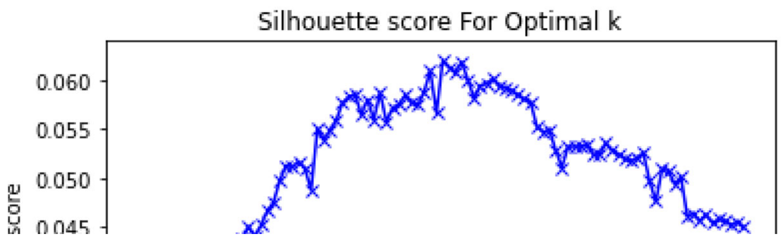
1. Fit K-Means clustering model to the text vector. What is the value of K you should pick? Why?
2. Use Silhouette score to evaluate your clusters. Try to evaluate the model for different values of k to see which k fits best for the dataset.

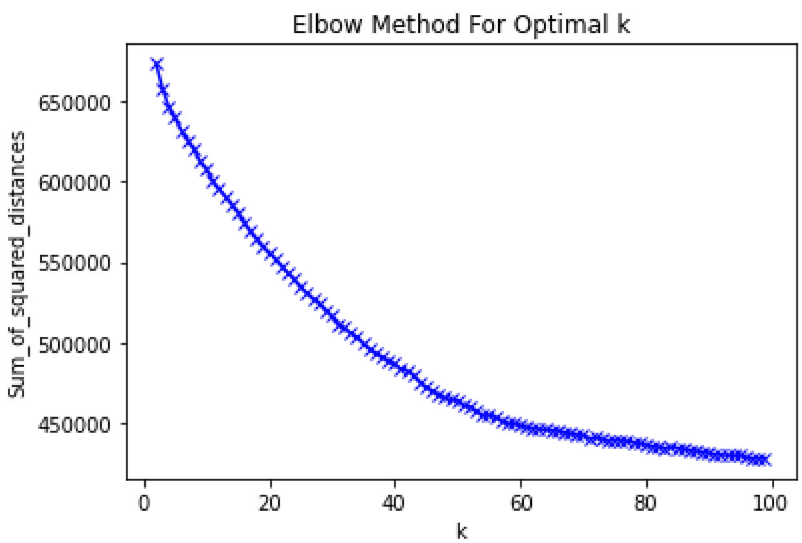
14
```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler
import pandas as pd
from matplotlib import pyplot as plt

X = pd.DataFrame(vectors.toarray(), columns=vectorizer.get_feature_names())
y = newsgroups.target
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

นำมา loop ลองค่า K ต่าง ๆ แล้วคำนวนหาทั้ง silhouette score และ sum od squared distance เพื่อนำมา plot graph ดูว่าควรใช้ค่า K เท่าไหร่ดีจาก elbow method

15
```
# Ref: https://blog.cambridgespark.com/how-to-determine-the-optimal-number-of-clusters-for-k-means-cluste
silhouette_score_list = []
sse = []
K = range(2, 100)
for k in K:
    kmeans = KMeans(init="random",n_clusters=k, n_init=10,max_iter=300, random_state=136)
    kmeans.fit(X)
    score = silhouette_score(X, kmeans.labels_)
    silhouette_score_list.append(score)
    sse.append(kmeans.inertia_)
    print("k=%d: silhouette score=%f"%(k, score), end='\r')
plt.plot(K, silhouette_score_list, 'bx-')
plt.xlabel('k')
plt.ylabel('silhouette score')
plt.title('Silhouette score For Optimal k')
plt.show()
plt.plot(K, sse, 'bx-')
plt.xlabel('k')
plt.ylabel('Sum_of_squared_distances')
plt.title('Elbow Method For Optimal k')
plt.show()
```





graph ของ silhouette score ดูเพื่อเลือกค่า K ที่ดีที่สุดได้ง่ายคือหยิบอันที่ได้ score มากที่สุดมาเลย แต่ graph ของ sum of squared error ดูยากจึงยึดตาม silhouette score ดีกว่า

16
```
best = silhouette_score_list.index([max(silhouette_score_list)])+2
print("Best k is %d (selected from highest silhouette score)"%best)
```

```
Best k is 51 (selected from highest silhouette score)
```

3.3 Topic Terms

We want to explore each cluster to understand what news articles are in the cluster, what terms are associated with the cluster. This will require a bit of hacking.

1. Use TfidfVectorizer.get feature names to extract words associated with each dimension of the text vector.
2. Extract cluster’s centroids using kmeans.cluster centers .

3. For each centroid, print the top 15 words that have the highest frequency.

นำค่า K ที่ทดสอบแลวว่าดีที่สุดมาทำ clustering แล้วดึง centroid จากนั้นทำการเรียงลำดับ tf-idf แล้ว print features ที่มี tf-idf มากที่สุด 15 อันดับแรกออกมา

17
```
kmeans = KMeans(init="random",n_clusters=best, n_init=10,max_iter=300)
kmeans.fit(X)
y_pred = kmeans.predict(X)
```

18
```
feature_names = np.array(vectorizer.get_feature_names()) # Get feature names
centroids = kmeans.cluster_centers_ # Get cluster's centroid
for i, centroid in enumerate(centroids):
    print("Centroid: %d"%i)
    tfidf_sorting = np.argsort(np.array(centroid)).flatten()[::-1] # Sorting tf-idf in centroid
    top_n = feature_names[tfidf_sorting][:15] # Get 15 word with top frequency
    print(top_n)
```

```
Centroid: 0
['sure' 'writes' 'article' 'edu' 'nntp' 'just' 'host' 'posting' 'make'
 'com' 'let' 'like' 'way' 'did' 'world']
Centroid: 1
['read' 'don' 'question' 'thanks' 'want' 'reply' 'know' 'need' 'help'
 'people' 'writes' 'say' 'work' 'use' 'posting']
Centroid: 2
['mail' 'thanks' 'reply' 'university' 've' 'know' 'read' 'used' 'new'
 'usa' 'use' 'like' 'does' 'll' 'ca']
Centroid: 3
['right' 'way' 'just' 'people' 'say' 'does' 'writes' 'did' 'years' 'like'
 'state' 'got' 'article' 'thing' 'said']
Centroid: 4
['cs' 'edu' 'computer' 'article' 'writes' 'university' 'reply' 'ca'
 'believe' 'doesn' 'distribution' 'thing' 'll' 'read' 'years']
Centroid: 5
['host' 'nntp' 'posting' 'thanks' 'reply' 'com' 'mail' '10' 'edu'
 'computer' 'need' 'does' 'help' 'know' 'used']
Centroid: 6
['usa' 'distribution' 'nntp' 'host' 'posting' 'edu' 'university' 'thanks'
 'mail' 'com' '10' 'computer' 'need' 'reply' 'work']
Centroid: 7
['doesn' 'like' 'just' 'know' 'writes' 'article' 'going' 'really' 'work'
 'does' 'thing' 'world' 'way' 'problem' 'say']
Centroid: 8
['years' 'writes' 'new' 'time' '10' 'article' 'question' 'people' 'com'
 'make' 'good' 'case' 'say' 'said' 'long']
Centroid: 9
['good' 'got' 'like' 'just' 'really' 'sure' 'better' 'things' 've' 'com'
 'going' 'thing' 'want' 'read' 'make']
Centroid: 10
['ll' 'mail' 'just' 'got' 'usa' 'like' 've' 'way' 'let' 'new' 'need'
 'really' 'distribution' 'don' 'know']
Centroid: 11
['use' 'used' 'using' 'need' 'work' 'want' 'like' 'way' 'better' 'does'
 'sure' 'help' 'make' 'don' 'know']
Centroid: 12
['help' 'thanks' 'need' 'mail' 'does' 'university' 'know' 'use' 'work'
 'computer' 'reply' 'problem' 'got' 'want' 'really']
Centroid: 13
['world' 'distribution' 'nntp' 'host' 'posting' 'reply' 'thanks' 'com'
 'help' '10' 'usa' 'edu' 'work' 'university' 'mail']
Centroid: 14
['people' 'things' 'say' 'just' 'believe' 'said' 'right' 'don' 'did'
```

```
 'world' 'make' 'point' 'think' 'like' 'way']
Centroid: 15
['said' 'people' 'news' 'say' 'did' 'going' 'work' 'time' 'like' 'think'
 '10' 'got' 'just' 'long' 'know']
Centroid: 16
['make' 'just' 'like' 'does' 'people' 'want' 'really' 'sure' 'don' 'got'
 'new' 'question' 'case' 'writes' 'article']
Centroid: 17
['work' 'use' 'doesn' 'way' 'does' 'need' 'problem' 've' 'better'
 'article' 'long' 'want' 'make' 'edu' 'just']
Centroid: 18
['long' 'time' 'years' 'work' 'make' 'better' 'really' 'like' 'just'
 'right' 'way' 'writes' 'com' 'point' 'good']
Centroid: 19
['edu' 'university' 'nntp' 'host' 'posting' 'reply' 'thanks' 'state' 'new'
 'mail' 'need' 'got' 'distribution' 'help' '10']
Centroid: 20
['ca' 'university' 'thanks' 'writes' 'posting' 'nntp' 'host' 'article'
 'reply' 'got' 'used' 'mail' 'll' 've' 'distribution']
Centroid: 21
['let' 'know' 'mail' 'll' 'com' 've' 'say' 'thanks' 'got' 'writes' 'just'
 'said' 'did' 'way' 'time']
Centroid: 22
['want' 'don' 'just' 'use' 'writes' 'way' 'need' 'mail' 'better' 'know'
 'like' 'people' 'right' 'state' 'help']
Centroid: 23
['does' 'know' 'thanks' 'university' 'way' 'like' 've' 'ca' 'work'
 'question' 'use' 'com' 'problem' 'just' 'think']
Centroid: 24
['going' 'way' 'just' 'right' 'got' 'll' 'said' 'don' 'writes' 'people'
 'think' 'know' 'really' 'article' 'did']
Centroid: 25
['com' 'reply' 'writes' 'article' 'posting' 'nntp' 'host' 'said' 'believe'
 'distribution' 'world' 'ca' 'news' 'got' 'don']
Centroid: 26
['used' 'use' 'new' 'com' 'way' 'using' 'work' '10' 'computer'
 'distribution' 'know' 'make' 'question' 'mail' 'like']
Centroid: 27
['think' 'don' 'people' 'things' 'say' 'like' 'really' 'way' 'just'
 'writes' 'better' 'believe' 'good' 'did' 'point']
Centroid: 28
['things' 'like' 'people' 'way' 'believe' 'thing' 'don' 'make' 'just'
 'said' 'say' 'know' 'think' 'better' 'time']
Centroid: 29
['case' 'usa' 'nntp' 'posting' 'host' 'edu' 'university' 'reply' 'just'
 'article' 'say' 'used' 'distribution' 'state' 'don']
Centroid: 30
['10' 'new' 'reply' 'make' 'll' 'years' 'usa' 'got' 'mail' 'using' 'want'
 've' 'distribution' 'state' 'ca']
Centroid: 31
['believe' 'say' 'people' 'think' 'does' 'don' 'point' 'know' 'way' 'said'
 'doesn' 'going' 'question' 'right' 'did']
Centroid: 32
['state' 'university' 'edu' 'article' 'new' '10' 'used' 'host' 'nntp'
 'writes' 'usa' 'years' 'posting' 'news' 'distribution']
Centroid: 33
['article' 'writes' 'edu' 'com' 'news' 'university' 'way' 'think' 'like'
 'just' 'state' 'did' 'ca' 'got' 'said']
Centroid: 34
['problem' 'using' 'help' 'thanks' 'computer' 'use' 'used' 'work' 'got'
 'just' 'time' 'know' 'need' 'don' 've']
Centroid: 35
['question' 'thanks' 'does' 'way' 'reply' 'did' 'time' 'article' 'problem'
 'don' 'computer' 'really' 'used' 'say' 'want']
Centroid: 36
```

```
['did' 'know' 'writes' 'said' 'got' 'say' 'make' 'way' 'let' 'case'
 'years' 'point' 'think' 'people' 'right']
Centroid: 37
['news' 'edu' 'host' 'nntp' 'posting' 'mail' 'university' 'article'
 'writes' 'world' 'reply' 'cs' 'said' 'thanks' 'distribution']
Centroid: 38
['say' 'did' 'way' 'people' 'think' 'just' 'point' 'don' 'doesn' 'writes'
 'years' 'know' 'believe' 'things' 'does']
Centroid: 39
['thing' 'way' 'just' 'good' 'did' 'writes' 'article' 'don' 'like' 'said'
 'posting' 'nntp' 'long' 'used' 'host']
Centroid: 40
['point' 'com' 'case' 'believe' 'question' 'time' 'think' 'really' 'does'
 'way' 'used' 'good' 'right' 'just' 'did']
Centroid: 41
['new' 'university' '10' 'distribution' 'state' 'news' 'years' 'used'
 'got' 'like' 'long' 'said' 'edu' 'mail' 'thanks']
Centroid: 42
['computer' 'university' 'thanks' 'mail' 'usa' 'use' 'read' 'distribution'
 'make' 'reply' 'cs' 'help' 'need' '10' 'know']
Centroid: 43
['using' 'use' 'help' 'want' 'does' 'need' 'thanks' 'used' 'work' 've'
 'problem' 'time' 'like' 'distribution' 'make']
Centroid: 44
['need' 'thanks' 'help' '10' 'want' 'know' 'computer' 'got' 'com' 'work'
 'going' 'sure' 'think' 'make' 'like']
Centroid: 45
['ve' 'got' 'just' 'going' 'problem' 'don' 'help' 'years' 'good' 'thanks'
 'way' 'like' 'used' 'time' 'thing']
Centroid: 46
['better' 'good' 'think' 'ca' 'article' 'way' 'writes' 'point' 'right'
 'say' 'going' 'really' 'years' 'don' 'just']
Centroid: 47
['using' 'host' 'nntp' 'posting' 'problem' 'edu' 'distribution' 'use'
 'help' 'world' 've' 'used' 'usa' 'question' 'ca']
Centroid: 48
['really' 'think' 'like' 'did' 'writes' 'want' 'just' 'good' 'does' 'don'
 'need' 've' 'sure' 'article' 'thing']
Centroid: 49
['don' 'know' 'like' 'just' 'way' 'people' 'good' 'going' 'think' 'doesn'
 'things' 'say' 'sure' 'll' 'want']
Centroid: 50
['time' 'just' 'like' 'long' 'think' 'read' 'said' 'problem' 'did' 'make'
 'world' 'years' 'got' 'really' 'way']
```