**Suppawich Tawornpichayachai 6231364021 Assignment 1 k-Nearest Neighbors**

**Note** ใน assignment นี้จะทำ data preprocessing ก่อนที่จะ split data (ข้อ 2)

1. Construct the kNN classifier to predict Breast Cancer (class 2 = benign, 4 = malignant)

สำหรับในส่วนแรกได้ทำการ import data ที่เป็น csv โดยดึงมาจาก Github และหลังจากนั้นก็ print ข้อมูลมาดูเป็นตัวอย่าง 5 records และได้ check shape ของ data พบว่ามีทั้งหมด 699 rows 11 columns

Import dataset

```
[2] df = pd.read_csv('https://raw.githubusercontent.com/upsaga77/data-mining/main/dataset_kNN.csv')
```

```
[3] df.head(5)
```

| | id | clump_thickness | size_uniformity | shape_uniformity | marginal_adhesion | epithelial_size | bare_nucleoli | bland_chromatin | normal_nucleoli | mitoses | class |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 188336 | 5 | 3 | 2 | 8 | 5 | 10 | 8 | 1 | 2 | 4 |
| 1 | 352431 | 10 | 5 | 10 | 3 | 5 | 8 | 7 | 8 | 3 | 4 |
| 2 | 353098 | 4 | 1 | 1 | 2 | 2 | 1 | 1 | 1 | 1 | 2 |
| 3 | 411359 | 1 | 1 | 1 | 1 | 2 | 1 | 1 | 1 | 1 | 2 |
| 4 | 411453 | 5 | 1 | 1 | 1 | 2 | 1 | 3 | 1 | 1 | 2 |

```
df.shape
```

```
(699, 11)
```

2. Split data into 80% training and 20% testing set (random_state = 1234 for reproducibility)

ทำการ split data หลังจากการทำ data preprocessing โดย y คือข้อมูลใน column “class” ซึ่งเป็น target และ X คือข้อมูลใน column ที่เหลือที่ผ่านการทำ data preprocessing เป็นที่เรียบร้อย

Split Data

```
from sklearn.model_selection import train_test_split

# Spilt the data into features and label
y = df_target
X = std_df
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.8, test_size = 0.2, random_state = 1234)
```

3. Data Preprocessing:

a. Convert the classes to a 0 (benign) and 1 (malignant) indicator for using in the classifier

ใช้คำสั่ง where ของ numpy โดยถ้าตอนแรก class = 2 (benign) จะถูกเปลี่ยนเป็น class 0 นอกจากนั้นจะถูกเปลี่ยนเป็น class 1 (malignant)

Convert the classes to a 0 (benign) and 1 (malignant) indicator for using in the classifier

```
[5] df["class"] = np.where(df["class"] == 2, 0, 1)
```

```
df.head(5)
```

| | id | clump_thickness | size_uniformity | shape_uniformity | marginal_adhesion | epithelial_size | bare_nucleoli | bland_chromatin | normal_nucleoli | mitoses | class |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 188336 | 5 | 3 | 2 | 8 | 5 | 10 | 8 | 1 | 2 | 1 |
| 1 | 352431 | 10 | 5 | 10 | 3 | 5 | 8 | 7 | 8 | 3 | 1 |
| 2 | 353098 | 4 | 1 | 1 | 2 | 2 | 1 | 1 | 1 | 1 | 0 |
| 3 | 411359 | 1 | 1 | 1 | 1 | 2 | 1 | 1 | 1 | 1 | 0 |
| 4 | 411453 | 5 | 1 | 1 | 1 | 2 | 1 | 3 | 1 | 1 | 0 |

b. Fill in Missing values, if exist (try using Mode value)

Missing values มีใน column “bare_nucleoli” เท่านั้นแต่ค่า missing ใน column นี้มีค่าเป็น “?” จึงต้องเปลี่ยนให้มีค่าเป็น null และหลังจากนั้นก็ fill null value ด้วย mode ส่วนการใช้คำสั่ง .isnull().sum() เพื่อเป็นการตรวจสอบว่ามี data ที่เป็น null ทั้งหมดเท่าใด

Fill in Missing values, if exist (try using Mode value)

```
[45] df.loc[df['bare_nucleoli'] == "?", 'bare_nucleoli'] = np.NaN

     df['bare_nucleoli'].isnull().sum()

     16

[49] df['bare_nucleoli'].fillna(df['bare_nucleoli'].mode().iloc[0], inplace = True)

[50] df['bare_nucleoli'].isnull().sum()

     0
```

c. Drop non-value added variables

ขั้นตอนนี้เป็นการ drop column ที่ไม่มีผลต่อ model โดยได้ทำการ drop column “id” ออกและจะ drop column ที่มีข้อมูลที่ซ้ำกันมากกว่า 70% ของจำนวนข้อมูลทั้งหมด ซึ่งทำให้ column “mitoses” ถูก drop ออกไป รวมถึงมีการแยกข้อมูลของ column ที่ใช้ในการ train กับ column ที่เป็น target เพื่อไปทำ standardization ต่อไป

Drop non-value added variables

```
     df_target = pd.DataFrame(df['class'], columns = ['class'])
     df.drop(columns=['id','class'],inplace=True)

[87] for i in df.columns:
       if df[i].value_counts().max() > 0.7 * df.shape[0]:
         del df[i]
```

d. Standardization→sklearn.preprocessing.StandardScaler

ขั้นตอนนี้เป็นการทำ standardization ซึ่งเป็นการปรับช่วงของข้อมูลให้อยู่ในช่วงที่กำหนด

```
[90] # Import standardization tool
     from sklearn.preprocessing import StandardScaler

     # Apply standardization
     scaler = StandardScaler()
     scaled = scaler.fit_transform(df)
     std_df = pd.DataFrame(scaled, columns=df.columns)

     std_df.head(5)
```

| | clump_thickness | size_uniformity | shape_uniformity | marginal_adhesion | epithelial_size | bare_nucleoli | bland_chromatin | normal_nucleoli |
|---|---|---|---|---|---|---|---|---|
| **0** | 0.206936 | -0.044102 | -0.406574 | 1.820022 | 0.806239 | 1.799664 | 1.872361 | -0.611825 |
| **1** | 1.983939 | 0.611792 | 2.287222 | 0.067687 | 0.806239 | 1.247077 | 1.461957 | 1.682167 |
| **2** | -0.148465 | -0.699995 | -0.743299 | -0.282780 | -0.549561 | -0.686979 | -1.000471 | -0.611825 |
| **3** | -1.214667 | -0.699995 | -0.743299 | -0.633247 | -0.549561 | -0.686979 | -1.000471 | -0.611825 |
| **4** | 0.206936 | -0.699995 | -0.743299 | -0.633247 | -0.549561 | -0.686979 | -0.179662 | -0.611825 |

4. Construct KNeighborsClassifier from sklearn.neighbors

ทำการ import libraries เพื่อทำ k-Nearest Neighbors และ metrics เพื่อแสดงผลลัพธ์ของ model ในรูปแบบของ accuracy score, classification report และ confusion matrix

## Construct KNeighborsClassifier

```
[19] from sklearn.neighbors import KNeighborsClassifier
     from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

5. Use Euclidean distance (Minkowski with 2-norm)

6. Manually tuning of k value with train-validate(); trying k = {1,2,3,4,5,6,7,8,9,10} and plotting accuracies over varied k-values. Train the model using the value of k yielding the highest accuracy on the validation set. Report the accuracy of the constructed model on the test set.

7. Repeat step 6, but tuning the value of k using GridSearchCV().

8. Submit the report PDF showing the confusion matrix of each constructed model with the selected k, and comparing the results for all points above with sound arguments.

Manually tuning of k value -> หลักการคือทำการ train model โดยไล่ค่า k ตั้งแต่ 1 จนถึง 10 โดยจะเก็บค่า accuracy ของแต่ละค่า k ไว้ใน list เพื่อนำไป plot graph และผลลัพธ์ออกมาคือได้ accuracy มากที่สุดเมื่อ k = 3 โดยมีค่า accuracy อยู่ที่ 0.9857

Manually tuning of k value

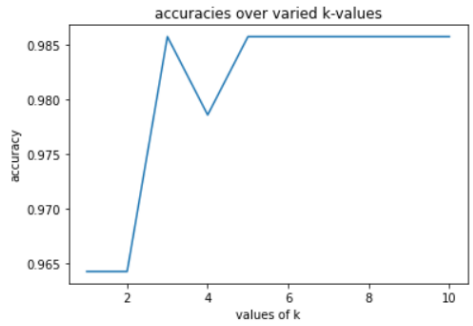
```
n_neighbors = [1,2,3,4,5,6,7,8,9,10]
accuracy = []
for i in n_neighbors:
  knn = KNeighborsClassifier(n_neighbors = i,weights = "distance", metric='minkowski')
  knn.fit(X_train, np.ravel(y_train))
  # Predict as Class
  accuracy.append(accuracy_score(y_test, knn.predict(X_test)))
max_acc = max(accuracy)
max_index = accuracy.index(max_acc)
print(max_acc,"is occured when k =", n_neighbors[max_index])
```

```
0.9857142857142858 is occured when k = 3
```

หลังจากที่เก็บค่า accuracy ตั้งแต่ k = 1 จนถึง k = 10 ไว้ใน list ก็นำมาแสดงผลเป็นกราฟเส้นภาพดังภาพด้านล่าง พบว่าที่ k = 3 ให้ค่า accuracy มากที่สุด

plot accuracy

```
plt.plot(n_neighbors, accuracy)
plt.title('accuracies over varied k-values')
plt.xlabel('values of k')
plt.ylabel('accuracy')
plt.show()
```




จากนั้นก็นำข้อมูลไป test โดยใช้ชุดข้อมูล test และสิ่งที่ model predict ได้ โดยใน classification report มีค่า accuracy เช่นเดียวกัน ส่วน confusion matrix สามารถบอกได้ว่า จากข้อมูล class 0 ทั้งหมด 95 ตัวอย่างพบว่าสามารถทำนายได้ถูกต้องว่าเป็น class 0 94 ตัวอย่างและทำนายผิดว่าเป็น class 1 จำนวน 1 ตัวอย่าง และในทำนองเดียวกัน จากข้อมูล class 1 ทั้งหมด 45 ตัวอย่างพบว่าสามารถทำนายได้ถูกต้องว่าเป็น class 1 44 ตัวอย่างและทำนายผิดเป็น class 0 จำนวน 1 ตัวอย่าง

```
[26] knn = KNeighborsClassifier(n_neighbors = n_neighbors[max_index],weights = "distance", metric='minkowski')
     knn.fit(X_train, np.ravel(y_train))
     print("Classification Report: \n {}\n".format(classification_report(y_test, knn.predict(X_test), digits=4)))
```

```
Classification Report:
              precision    recall  f1-score   support

           0     0.9895    0.9895    0.9895        95
           1     0.9778    0.9778    0.9778        45

    accuracy                         0.9857       140
   macro avg     0.9836    0.9836    0.9836       140
weighted avg     0.9857    0.9857    0.9857       140
```

```
print("Confusion Matrix: \n {}\n".format(confusion_matrix(y_test, knn.predict(X_test))))
```

```
Confusion Matrix:
 [[94  1]
 [ 1 44]]
```

ต่อมาก็ใช้ grid search เพื่อหาค่า k ที่ดีที่สุดสำหรับ model นี้โดยเริ่มจาก grid_params เป็นการกำหนดค่า parameter ของ model และหลังจากนั้นก็ใช้คำสั่ง knn_w_grid.best_params_ เพื่อหาค่า parameter ที่ดีที่สุด พบว่าได้ค่า k = 9 และ weight เป็น uniform (น้ำหนักเท่ากัน) ซึ่งต่างกับแบบ distance ที่อยู่ใกล้จะมีสัดส่วนน้ำหนักมากกว่าส่วนที่อยู่ไกล

using grid search

```
from sklearn.model_selection import GridSearchCV
grid_params = {
    'n_neighbors': [1,2,3,4,5,6,7,8,9,10],
    'weights': ['uniform','distance'],
    'metric': ['minkowski']
}
knn_w_grid = GridSearchCV(
    KNeighborsClassifier(),
    grid_params,
    verbose = 1,
    cv = 3,
    n_jobs = -1
)
knn_w_grid.fit(X_train, np.ravel(y_train))
knn_w_grid.best_params_
```

```
Fitting 3 folds for each of 20 candidates, totalling 60 fits
{'metric': 'minkowski', 'n_neighbors': 9, 'weights': 'uniform'}
```

ผลลัพธ์ที่ได้จาก grid search ซึ่งสามารถดูได้จาก classification report และ confusion matrix พบว่าได้ผลลัพธ์เหมือนกับการหาค่า k ด้วยวิธีการด้านบน (สังเกตจาก confusion matrix ได้ค่าเดียวกับ matrix ทางด้านบน) โดยผมคิดว่าสาเหตุที่ทั้ง 2 วิธีให้ค่า accuracy ไม่แตกต่างจากกันเพราะว่า dataset ในโจทย์นี้ค่อนข้างเล็กครับ

```
knn = KNeighborsClassifier(n_neighbors = best_params['n_neighbors'],weights = best_params['weights'], metric='minkowski')
knn.fit(X_train, np.ravel(y_train))
print("Classification Report: \n {}\n".format(classification_report(y_test, knn.predict(X_test), digits=4)))
print("Confusion Matrix: \n {}\n".format(confusion_matrix(y_test, knn.predict(X_test))))
```

```
Classification Report:
              precision    recall  f1-score   support

           0     0.9895    0.9895    0.9895        95
           1     0.9778    0.9778    0.9778        45

    accuracy                         0.9857       140
   macro avg     0.9836    0.9836    0.9836       140
weighted avg     0.9857    0.9857    0.9857       140


Confusion Matrix:
 [[94  1]
 [ 1 44]]
```

9. Include the link to your colab notebook.

https://colab.research.google.com/drive/1XPLmcmVhPGHIsxy103sg6FFhwsfT4Rf2?usp=sharing