Bank Marketing Campaigns Dataset Analysis

K-Nearest Neighbor (KNN) Algorithm

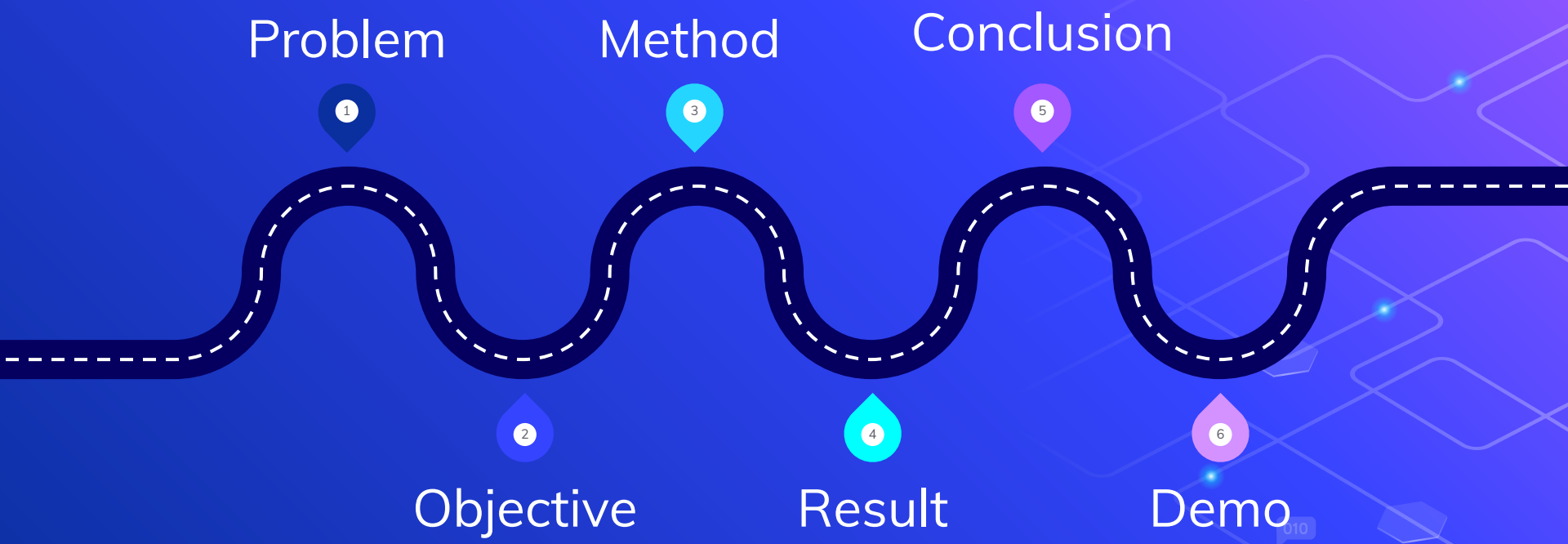
Team Members : STRONG!

Muhammad Naufal Hakim
19/440307/TK/48634
[Programmer]

Samatha Marhaendra Putra
19/444071/TK/49267
[Programmer & Editor]

Lathif Ma'arif
19/444058/TK/49254
[Editor]

Roadmap



Problem Definition

Problem Definition

Objective

Give **prediction** whether client will **subscribe** bank deposit **or not**.

Dataset Characteristic

Bank Marketing
Campaign Analysis

21 Features

age, job, marital, education, default, housing, loan, contact, month, day_of_week, duration, campaign, pdays, previous, poutcome, emp.var.rate, cons.price.idx, cons.conf.idx, euribor3m, nr.employed, y

41,188 Row Data

Divided to Train, Validate and Test Data

Yes / No

Prediction Result

Method

K-Nearest Neighbor (KNN)

Algorithm

Why Using KNN Algorithm

Calculation Time

This dataset has a lot of row, but we process it using limited device capatity. Therefore, we need an efficient algorithm, one of the example is KNN algorithm.

Prediction Power

This algorithm will be used in banking industry, which need user satisfaction. This prediction power of the algorithm will increase user satisfaction in banking industry.

Import Library & Dataset

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')

import math
from collections import Counter
import operator

df = pd.read_csv('bank-additional-full.csv',delimiter=";")
```

Slicing Features into Dataframes

```
bank_client = df.iloc[: , 0:7]
bank_related = df.iloc[: , 7:11]
bank_se = df.loc[: , ['emp.var.rate', 'cons.price.idx',
                      'cons.conf.idx', 'euribor3m', 'nr.employed']]
bank_o = df.loc[: , ['campaign', 'pdays','previous', 'poutcome']]
```

Data Pre-Processing - 1

```
bank_client['job'].replace(['housemaid' , 'services' , 'admin.' , 'blue-collar' , 'technician', 'retired' ,
'management', 'unemployed', 'self-employed', 'unknown' , 'entrepreneur', 'student'] , [1, 2, 3, 4, 5, 6, 7, 8, 9,
10, 11, 12], inplace=True)
bank_client['education'].replace(['basic.4y' , 'high.school', 'basic.6y', 'basic.9y', 'professional.course',
'unknown' , 'university.degree' , 'illiterate'], [1, 2, 3, 4, 5, 6, 7, 8], inplace=True)
bank_client['marital'].replace(['married', 'single', 'divorced', 'unknown'], [1, 2, 3, 4], inplace=True)
bank_client['default'].replace(['yes', 'no', 'unknown'],[1, 2, 3], inplace=True)
bank_client['housing'].replace(['yes', 'no', 'unknown'],[1, 2, 3], inplace=True)
bank_client['loan'].replace(['yes', 'no', 'unknown'],[1, 2, 3], inplace=True)

def age(dataframe):
    dataframe.loc[dataframe['age'] ⩽ 32, 'age'] = 1
    dataframe.loc[(dataframe['age'] > 32) & (dataframe['age'] ⩽ 47), 'age'] = 2
    dataframe.loc[(dataframe['age'] > 47) & (dataframe['age'] ⩽ 70), 'age'] = 3
    dataframe.loc[(dataframe['age'] > 70) & (dataframe['age'] ⩽ 98), 'age'] = 4

    return dataframe

age(bank_client);
```

Data Pre-Processing - 2

```
bank_related['contact'].replace(['telephone', 'cellular'], [1, 2], inplace=True)
bank_related['month'].replace(['may', 'jun', 'jul', 'aug', 'oct', 'nov', 'dec', 'mar', 'apr', 'sep'], [1, 2, 3, 4,
5, 6, 7, 8, 9, 10], inplace=True)
bank_related['day_of_week'].replace(['mon', 'tue', 'wed', 'thu', 'fri'], [1, 2, 3, 4, 5], inplace=True)

def duration(data):
    data.loc[data['duration'] ⩽ 102, 'duration'] = 1
    data.loc[(data['duration'] > 102) & (data['duration'] ⩽ 180)  , 'duration']    = 2
    data.loc[(data['duration'] > 180) & (data['duration'] ⩽ 319)  , 'duration']   = 3
    data.loc[(data['duration'] > 319) & (data['duration'] ⩽ 644.5), 'duration'] = 4
    data.loc[data['duration']  > 644.5, 'duration'] = 5
    return data
duration(bank_related);
```

Data Pre-Processing - 3

```
bank_o['poutcome'].replace(['nonexistent', 'failure', 'success'], [1,2,3], inplace  = True)
```

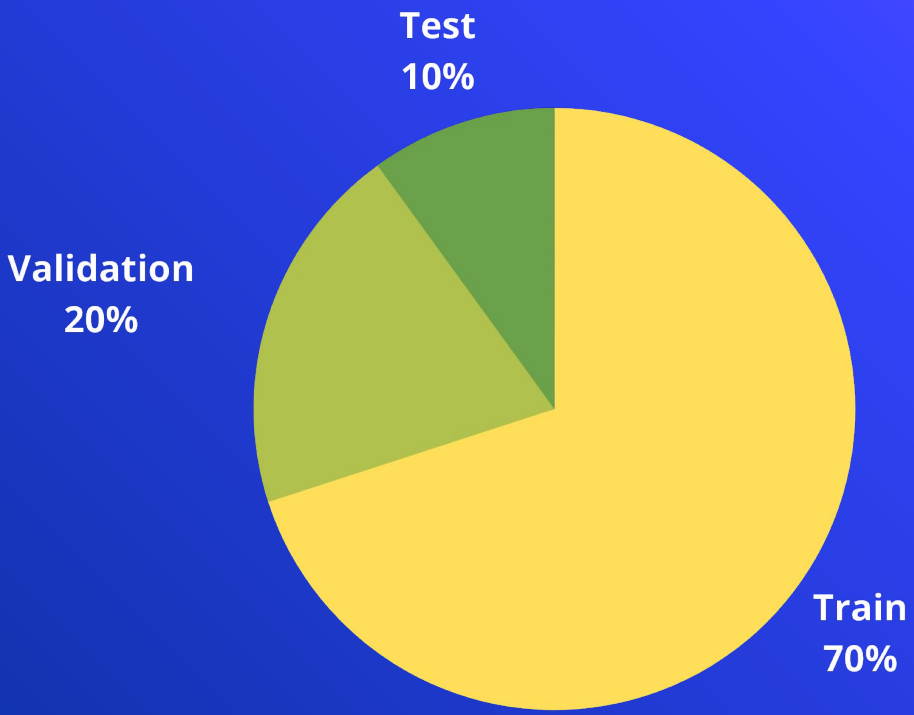
Data Post-Processing

```
bank_final= pd.concat([bank_client, bank_related, bank_se, bank_o, df['y']], axis = 1)
bank_final = bank_final[['age', 'job', 'marital', 'education', 'default', 'housing',
                         'loan','contact', 'month', 'day_of_week', 'duration',
                         'emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euribor3m',
                         'nr.employed', 'campaign', 'pdays', 'previous','poutcome','y']]
```

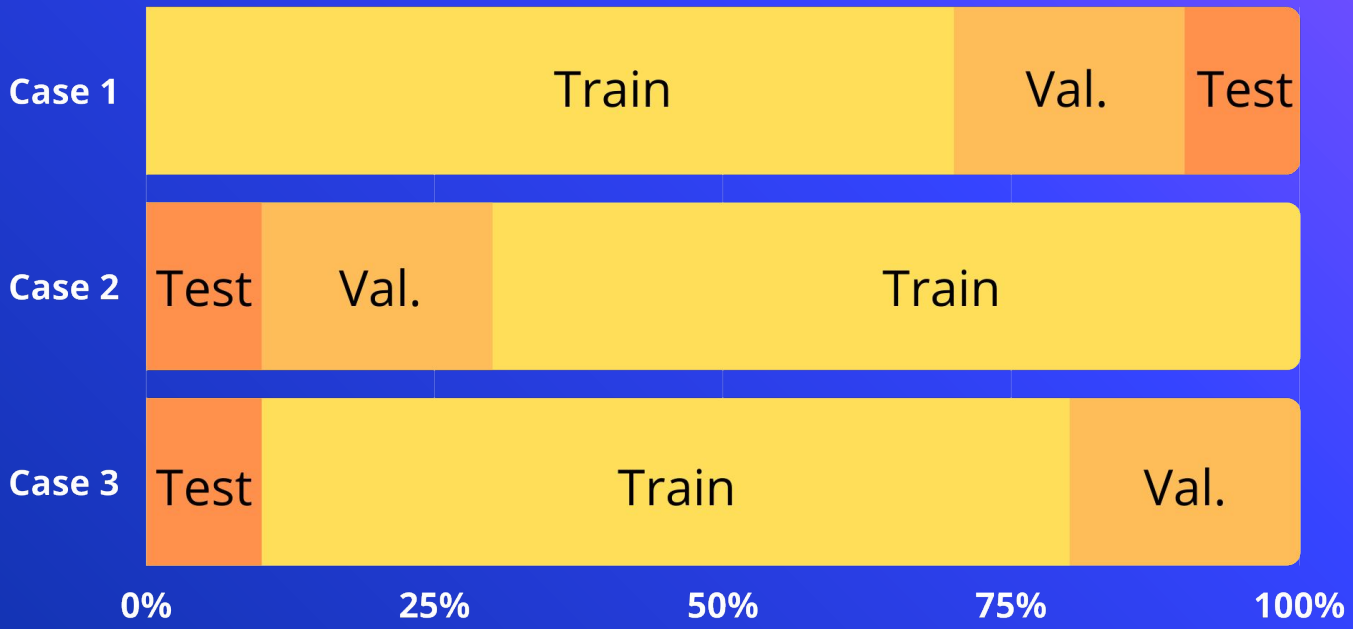
```
bank_final['y'].replace(['no', 'yes'], [0,1], inplace  = True)
```

KNN Preparation

Train, Validate and Test Data



Train - Validate - Test Data



Train - Validate - Test Case 1

```
length_train_low = int(bank_final_random.shape[0]*0)
length_train_up = int(bank_final_random.shape[0]*0.7)

length_val_low = int(bank_final_random.shape[0]*0.7)
length_val_up = int(bank_final_random.shape[0]*0.9)

length_test_low = int(bank_final_random.shape[0]*0.9)
length_test_up = int(bank_final_random.shape[0]*1)
```

Train - Validate - Test Case 2

```
length_train_low = int(bank_final_random.shape[0]*0.3)
length_train_up = int(bank_final_random.shape[0]*1)

length_val_low = int(bank_final_random.shape[0]*0.1)
length_val_up = int(bank_final_random.shape[0]*0.3)

length_test_low = int(bank_final_random.shape[0]*0)
length_test_up = int(bank_final_random.shape[0]*0.1)
```

Train - Validate - Test Case 3

```
length_train_low = int(bank_final_random.shape[0]*0.1)
length_train_up = int(bank_final_random.shape[0]*0.8)

length_val_low = int(bank_final_random.shape[0]*0.8)
length_val_up = int(bank_final_random.shape[0]*1)

length_test_low = int(bank_final_random.shape[0]*0)
length_test_up = int(bank_final_random.shape[0]*0.1)
```

Train - Validate - Test Data

```
train_bank_final = bank_final_random.iloc
                   [length_train_low:length_train_up,:]
validate_bank_final = bank_final_random.iloc
                   [length_val_low:length_val_up,:]
test_bank_final = bank_final_random.iloc
                   [length_test_low:length_test_up,:]
                   .reset_index().drop(["index"],axis=1)
```

Train
Validate
Test

Set

```
train_set={0:[],1:[]}
test_set={0: [],1:[]}
validate_set={0:[],1: []}
for i in train_bank_final:
  train_set[i[-1]].append(i[:-1])
# appending list to last element
for i in test_bank_final:
  test_set[i[-1]].append(i[:-1])
  # appending list to last element
for i in validate_bank_final:
  validate_set[i[-1]].append(i[:-1])
  # appending list to last element
```

KNN Algorithm

KNN Algorithm

```
def k_nearest_neighbors(data,predict,k):
  distances=[]
  for group in data:
    for features in data[group]:
      euclidean_distance = math.sqrt((features[0]-predict[0])**2
                          + (features[1]-predict[1])**2)
      distances.append([euclidean_distance, group])
  votes=[i[1] for i in sorted(distances)[:k]]
  vote_result=Counter(votes).most_common(1)[0][0]
  return vote_result
```

Training Algorithm

```
k_range=range(1,5)
correct = 0
total = 0
max_accuracy = 0.0
optimal_k = 0
accuracy = 0.0
accuracies = []
for k in k_range:
  for group in validate_set:
    for data in validate_set[group]:
      vote=k_nearest_neighbors(train_set,data,k)
      if group==vote:
        correct+=1
      total+=1
      accuracy = correct/total
  print("Accuracy with ", k, accuracy)
```

Result

Finding Optimal K

```
if max_accuracy < accuracy:
  max_accuracy = accuracy
  optimal_k = k
accuracies.append(accuracy)

print('max accuracy: ', max_accuracy)
print('optimal k: ', optimal_k)
```

Testing Model

```
for group in test_set:
    for data in test_set[group]:
        vote=k_nearest_neighbors
        (test_set,data,optimal_k)
        if group==vote:
            correct+=1
        total+=1
        accuracy=correct/total
print('Accuracy: ', accuracy)
```

Cross Validation Result

Case 1

Nilai k	Accuracy
k = 1	**0.8897790725904345**
k = 2	0.8897790725904345
k = 3	0.8897386096949098
k = 4	0.8897487254187910

Case 2

Nilai k	Accuracy
k = 1	0.8828599174556931
k = 2	0.8828599174556931
k = 3	0.8829003803512180
k = 4	**0.8829206117989803**

Case 3

Nilai k	Accuracy
k = 1	0.8917212915756252
k = 2	0.8917212915756252
k = 3	**0.8917617544711499**
k = 4	0.8917516387472687

Conclusion

This algorithm used in this case can give 89,176% accuracy.

We can get this result by using k=3 and composition of data that we can see below:

Thanks!