# hw2-1

February 4, 2024

[33]:
```
import numpy as np
import scipy as sp
import pandas as pd
from IPython.display import display, HTML
df1=pd.read_csv("malware_Binary.csv")
```

[34]:
```
(df1.head())
```

[34]:

| | hash | millisecond | classification | os | state | usage_counter | prio | static_prio | normal_prio | policy | … | nivcsw | min_flt | maj_flt | fs_excl_counter | lock | utime | stime | gtime | cgtime | signal_nvcsw |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914… | 0 | malware | CentOS | 0 | 0 | 3069378560 | 14274 | 0 | 0 | … | 0 | 0 | 120 | 0 | 3204448256 | 380690 | 4 | 0 | 0 | 0 |
| 1 | 42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914… | 1 | malware | Windows | 0 | 0 | 3069378560 | 14274 | 0 | 0 | … | 0 | 0 | 120 | 0 | 3204448256 | 380690 | 4 | 0 | 0 | 0 |
| 2 | 42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914… | 2 | malware | Mac | 0 | 0 | 3069378560 | 14274 | 0 | 0 | … | 0 | 0 | 120 | 0 | 3204448256 | 380690 | 4 | 0 | 0 | 0 |
| 3 | 42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914… | 3 | malware | Ubuntu | 0 | 0 | 3069378560 | 14274 | 0 | 0 | … | 0 | 0 | 120 | 0 | 3204448256 | 380690 | 4 | 0 | 0 | 0 |
| 4 | 42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914… | 4 | malware | Mac | 0 | 0 | 3069378560 | 14274 | 0 | 0 | … | 0 | 0 | 120 | 0 | 3204448256 | 380690 | 4 | 0 | 0 | 0 |

[5 rows x 36 columns]

[35]:
```
df=df1.loc[:,['classification', 'os', 'usage_counter', 'prio', 'static_prio',␣
 ↪'normal_prio', 'vm_pgoff', 'vm_truncate_count', 'task_size', 'map_count',␣
 ↪'hiwater_rss', 'total_vm', 'shared_vm', 'exec_vm', 'reserved_vm', 'nr_ptes',␣
 ↪'nvcsw', 'nivcsw', 'signal_nvcsw']]
```

[36]:
```
df.head()
```

[36]:

| | classification | os | usage_counter | prio | static_prio | \ |
|---|---|---|---|---|---|---|
| 0 | malware | CentOS | 0 | 3069378560 | 14274 | |
| 1 | malware | Windows | 0 | 3069378560 | 14274 | |
| 2 | malware | Mac | 0 | 3069378560 | 14274 | |
| 3 | malware | Ubuntu | 0 | 3069378560 | 14274 | |
| 4 | malware | Mac | 0 | 3069378560 | 14274 | |

| | normal_prio | vm_pgoff | vm_truncate_count | task_size | map_count | \ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 13173 | 0 | 6850 | |
| 1 | 0 | 0 | 13173 | 0 | 6850 | |
| 2 | 0 | 0 | 13173 | 0 | 6850 | |
| 3 | 0 | 0 | 13173 | 0 | 6850 | |
| 4 | 0 | 0 | 13173 | 0 | 6850 | |

| | hiwater_rss | total_vm | shared_vm | exec_vm | reserved_vm | nr_ptes | nvcsw | \ |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 150 | 120 | 124 | 210 | 0 | 341974 | |
| 1 | 0 | 150 | 120 | 124 | 210 | 0 | 341974 | |
| 2 | 0 | 150 | 120 | 124 | 210 | 0 | 341974 | |
| 3 | 0 | 150 | 120 | 124 | 210 | 0 | 341974 | |
| 4 | 0 | 150 | 120 | 124 | 210 | 0 | 341974 | |

| | nivcsw | signal_nvcsw |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 2 | 0 | 0 |
| 3 | 0 | 0 |
| 4 | 0 | 0 |

[37]:
```
# Converting os and classification into integer variables
df_dummies=pd.get_dummies(df[['os']], dtype=float)
print(df_dummies.head())
```

| | os_CentOS | os_Debian | os_Mac | os_Ubuntu | os_Windows |
|---|---|---|---|---|---|
| 0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 1 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 |
| 2 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 |
| 3 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 |
| 4 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 |

[38]:
```
df = df.join(df_dummies)
df = df.drop('os', axis=1)
```

[39]:
```
df.head()
```

[39]:

| | classification | usage_counter | prio | static_prio | normal_prio | \ |
|---|---|---|---|---|---|---|
| 0 | malware | 0 | 3069378560 | 14274 | 0 | |
| 1 | malware | 0 | 3069378560 | 14274 | 0 | |
| 2 | malware | 0 | 3069378560 | 14274 | 0 | |
| 3 | malware | 0 | 3069378560 | 14274 | 0 | |
| 4 | malware | 0 | 3069378560 | 14274 | 0 | |

| | vm_pgoff | vm_truncate_count | task_size | map_count | hiwater_rss | … | \ |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 13173 | 0 | 6850 | 0 | … | |
| 1 | 0 | 13173 | 0 | 6850 | 0 | … | |
| 2 | 0 | 13173 | 0 | 6850 | 0 | … | |
| 3 | 0 | 13173 | 0 | 6850 | 0 | … | |
| 4 | 0 | 13173 | 0 | 6850 | 0 | … | |

| | reserved_vm | nr_ptes | nvcsw | nivcsw | signal_nvcsw | os_CentOS | os_Debian | \ |
|---|---|---|---|---|---|---|---|---|
| 0 | 210 | 0 | 341974 | 0 | 0 | 1.0 | 0.0 | |
| 1 | 210 | 0 | 341974 | 0 | 0 | 0.0 | 0.0 | |
| 2 | 210 | 0 | 341974 | 0 | 0 | 0.0 | 0.0 | |
| 3 | 210 | 0 | 341974 | 0 | 0 | 0.0 | 0.0 | |
| 4 | 210 | 0 | 341974 | 0 | 0 | 0.0 | 0.0 | |

| | os_Mac | os_Ubuntu | os_Windows |
|---|---|---|---|
| 0 | 0.0 | 0.0 | 0.0 |
| 1 | 0.0 | 0.0 | 1.0 |
| 2 | 1.0 | 0.0 | 0.0 |
| 3 | 0.0 | 1.0 | 0.0 |
| 4 | 1.0 | 0.0 | 0.0 |

[5 rows x 23 columns]

[40]:
```
df=df.drop('os_CentOS',axis=1) #Dropping os_centOS to maintain n-1 columns
```

[41]:
```
for col in df.columns:
    # removing classification as it is the label
    if col != 'classification':
        df[col]=(df[col]-df[col].min())/(df[col].max()-df[col].min())
        # both min and max value of the predictor task_size is zero so dropped.
    # it does not contribute to distance calculation
df = df.drop('task_size', axis= 1)
```

[42]:
```
df.isna().sum()
```

[42]:
```
classification            0
usage_counter        100000
prio                      0
static_prio               0
normal_prio          100000
vm_pgoff             100000
vm_truncate_count         0
map_count                 0
hiwater_rss          100000
total_vm                  0
shared_vm                 0
exec_vm                   0
reserved_vm               0
nr_ptes              100000
nvcsw                     0
nivcsw                    0
signal_nvcsw         100000
os_Debian                 0
os_Mac                    0
os_Ubuntu                 0
os_Windows                0
dtype: int64
```

[43]:
```
df.describe()
```

[43]:

| | usage_counter | prio | static_prio | normal_prio | vm_pgoff |
|---|---|---|---|---|---|
| count | 0.0 | 100000.000000 | 100000.000000 | 0.0 | 0.0 |
| mean | NaN | 0.499762 | 0.234841 | NaN | NaN |
| std | NaN | 0.287065 | 0.258006 | NaN | NaN |
| min | NaN | 0.000000 | 0.000000 | NaN | NaN |
| 25% | NaN | 0.248016 | 0.020373 | NaN | NaN |
| 50% | NaN | 0.492063 | 0.121509 | NaN | NaN |
| 75% | NaN | 0.743056 | 0.458611 | NaN | NaN |
| max | NaN | 1.000000 | 1.000000 | NaN | NaN |

| | vm_truncate_count | map_count | hiwater_rss | total_vm |
|---|---|---|---|---|
| count | 100000.000000 | 100000.000000 | 0.0 | 100000.000000 |
| mean | 0.321712 | 0.241567 | NaN | 0.093546 |
| std | 0.186489 | 0.147887 | NaN | 0.111189 |
| min | 0.000000 | 0.000000 | NaN | 0.000000 |
| 25% | 0.169110 | 0.150023 | NaN | 0.033856 |
| 50% | 0.317833 | 0.206165 | NaN | 0.061654 |
| 75% | 0.456305 | 0.316299 | NaN | 0.115110 |
| max | 1.000000 | 1.000000 | NaN | 1.000000 |

| | shared_vm | exec_vm | reserved_vm | nr_ptes | nvcsw |
|---|---|---|---|---|---|
| count | 100000.000000 | 100000.000000 | 100000.000000 | 0.0 | 100000.000000 |

```
mean        0.740030       0.343059       0.242872      NaN       0.226876
std         0.389611       0.214211       0.155259      NaN       0.194690
min         0.000000       0.000000       0.000000      NaN       0.000000
25%         0.250000       0.192308       0.114325      NaN       0.091519
50%         1.000000       0.336538       0.225895      NaN       0.204049
75%         1.000000       0.442308       0.336088      NaN       0.298492
max         1.000000       1.000000       1.000000      NaN       1.000000

              nivcsw  signal_nvcsw      os_Debian         os_Mac  \
count  100000.000000           0.0  100000.000000  100000.000000
mean        0.090387           NaN       0.199140       0.199460
std         0.144466           NaN       0.399356       0.399596
min         0.000000           NaN       0.000000       0.000000
25%         0.002740           NaN       0.000000       0.000000
50%         0.024658           NaN       0.000000       0.000000
75%         0.126027           NaN       0.000000       0.000000
max         1.000000           NaN       1.000000       1.000000

           os_Ubuntu     os_Windows
count  100000.000000  100000.000000
mean        0.200680       0.201100
std         0.400511       0.400825
min         0.000000       0.000000
25%         0.000000       0.000000
50%         0.000000       0.000000
75%         0.000000       0.000000
max         1.000000       1.000000
```

[44]:
```
# other predictors have max and min value 0 i.e all values are zero, they do␣
 ↪not contribute in calculating the distance, so dropped
df = df.drop(['usage_counter','normal_prio', 'vm_pgoff',\
              'hiwater_rss', 'nr_ptes', 'signal_nvcsw'], axis=1)
```

[45]:
```
# checking for null values in data
df.isna().sum()
```

[45]:
```
classification       0
prio                 0
static_prio          0
vm_truncate_count    0
map_count            0
total_vm             0
shared_vm            0
exec_vm              0
reserved_vm          0
nvcsw                0
nivcsw               0
```

```
os_Debian            0
os_Mac               0
os_Ubuntu            0
os_Windows           0
dtype: int64
```

[46]:
```
from sklearn import preprocessing
y = df['classification'] # define label as nominal values
le = preprocessing.LabelEncoder()
le.fit(y)
y_encoded = le.transform(y) # encode nominal labels to integers

print(y_encoded)

df['classification'] = y_encoded
x_features = df.drop('classification',axis=1)
```

```
[1 1 1 … 1 1 1]
```

[47]:
```
df.head()
```

[47]:

| | classification | prio | static_prio | vm_truncate_count | map_count | total_vm | shared_vm | exec_vm | reserved_vm | nvcsw | nivcsw | os_Debian | os_Mac | os_Ubuntu | os_Windows |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0.18254 | 0.016007 | 0.199175 | 0.16651 | 0.052031 | 1.0 | 0.307692 | 0.249311 | 0.091519 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 1 | 1 | 0.18254 | 0.016007 | 0.199175 | 0.16651 | 0.052031 | 1.0 | 0.307692 | 0.249311 | 0.091519 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 |
| 2 | 1 | 0.18254 | 0.016007 | 0.199175 | 0.16651 | 0.052031 | 1.0 | 0.307692 | 0.249311 | 0.091519 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 |
| 3 | 1 | 0.18254 | 0.016007 | 0.199175 | 0.16651 | 0.052031 | 1.0 | 0.307692 | 0.249311 | 0.091519 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 |
| 4 | 1 | 0.18254 | 0.016007 | 0.199175 | 0.16651 | 0.052031 | 1.0 | 0.307692 | 0.249311 | 0.091519 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 |

[48]:
```
df.head()
```

[48]:

| | classification | prio | static_prio | vm_truncate_count | map_count |
|---|---|---|---|---|---|
| 0 | 1 | 0.18254 | 0.016007 | 0.199175 | 0.16651 |
| 1 | 1 | 0.18254 | 0.016007 | 0.199175 | 0.16651 |

```
2                1  0.18254     0.016007           0.199175    0.16651
3                1  0.18254     0.016007           0.199175    0.16651
4                1  0.18254     0.016007           0.199175    0.16651

   total_vm  shared_vm   exec_vm  reserved_vm     nvcsw  nivcsw  os_Debian  \
0  0.052031        1.0  0.307692     0.249311  0.091519     0.0        0.0
1  0.052031        1.0  0.307692     0.249311  0.091519     0.0        0.0
2  0.052031        1.0  0.307692     0.249311  0.091519     0.0        0.0
3  0.052031        1.0  0.307692     0.249311  0.091519     0.0        0.0
4  0.052031        1.0  0.307692     0.249311  0.091519     0.0        0.0

   os_Mac  os_Ubuntu  os_Windows
0     0.0        0.0         0.0
1     0.0        0.0         1.0
2     1.0        0.0         0.0
3     0.0        1.0         0.0
4     1.0        0.0         0.0
```

[49]:
```
# splitting the data into train and test
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x_features, y_encoded,
                                                    test_size=0.2)
```

[50]:
```
#out of curiosity I also tried hold out evaluation
from sklearn import neighbors
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score

import warnings
warnings.filterwarnings("ignore")

# using both the metrics euclidean and manhattan distance
distances = ['euclidean','manhattan']
for i in distances:
    print('metric: ', i)
    # k in range 5 to 105 with stepsize 10
    for k in range(5,105,10):
        classifier=neighbors.KNeighborsClassifier(k, metric=i)
        classifier.fit(x_train, y_train)
        y_pred = classifier.predict(x_test)
        # Prints test accuracy, precision, recall and AUC
        print('K =', k, ', Accuracy: ', accuracy_score(y_test, y_pred), ',␣
 ↪Precision: ', precision_score(y_test, y_pred, average='macro',␣
 ↪zero_division=0),
```

```
          ', Recall: ', recall_score(y_test, y_pred, average='macro',␣
↪zero_division=0),', AUC:', roc_auc_score(y_test, y_pred))
```

```
metric:  euclidean
K = 5 , Accuracy:  0.99985 , Precision:  0.9998498799039232 , Recall:
0.9998501648186995 , AUC: 0.9998501648186995
K = 15 , Accuracy:  0.9999 , Precision:  0.9999001298312193 , Recall:
0.9998998898788667 , AUC: 0.9998998898788667
K = 25 , Accuracy:  0.99895 , Precision:  0.9989498793279228 , Recall:
0.998950163729698 , AUC: 0.998950163729698
K = 35 , Accuracy:  0.9966 , Precision:  0.9966000599660005 , Recall:
0.9966006558867937 , AUC: 0.9966006558867936
K = 45 , Accuracy:  0.9952 , Precision:  0.995201204193457 , Recall:
0.995201204193457 , AUC: 0.9952012041934571
K = 55 , Accuracy:  0.9927 , Precision:  0.9927087801832006 , Recall:
0.9927026311701836 , AUC: 0.9927026311701836
K = 65 , Accuracy:  0.99125 , Precision:  0.9912600601609626 , Recall:
0.9912527944158813 , AUC: 0.9912527944158813
K = 75 , Accuracy:  0.98865 , Precision:  0.9886873693387144 , Recall:
0.9886551012726725 , AUC: 0.9886551012726725
K = 85 , Accuracy:  0.9852 , Precision:  0.9852509922658623 , Recall:
0.9852059220991658 , AUC: 0.9852059220991658
K = 95 , Accuracy:  0.983 , Precision:  0.9830395746092655 , Recall:
0.9830052594363639 , AUC: 0.9830052594363639
metric:  manhattan
K = 5 , Accuracy:  1.0 , Precision:  1.0 , Recall:  1.0 , AUC: 1.0
K = 15 , Accuracy:  1.0 , Precision:  1.0 , Recall:  1.0 , AUC: 1.0
K = 25 , Accuracy:  0.99955 , Precision:  0.9995498498379459 , Recall:
0.999550274455832 , AUC: 0.9995502744558321
K = 35 , Accuracy:  0.99725 , Precision:  0.9972521529655116 , Recall:
0.9972514816742928 , AUC: 0.9972514816742927
K = 45 , Accuracy:  0.9964 , Precision:  0.9964038909765833 , Recall:
0.9964018656462574 , AUC: 0.9964018656462574
K = 55 , Accuracy:  0.99515 , Precision:  0.9951554220185086 , Recall:
0.9951521391340883 , AUC: 0.9951521391340883
K = 65 , Accuracy:  0.99325 , Precision:  0.9932547644022074 , Recall:
0.9932520268349525 , AUC: 0.9932520268349525
K = 75 , Accuracy:  0.99125 , Precision:  0.9912709857554872 , Recall:
0.9912538944172122 , AUC: 0.9912538944172122
K = 85 , Accuracy:  0.98885 , Precision:  0.988896440194698 , Recall:
0.9888556515153384 , AUC: 0.9888556515153383
K = 95 , Accuracy:  0.9877 , Precision:  0.9877492443693043 , Recall:
0.9877058151240363 , AUC: 0.9877058151240362
```

[51]:
```
# 10-Fold cross validation on the dataset using euclidean distance
from sklearn.model_selection import cross_val_score
```

```
k_val = [3, 5, 11, 31, 51, 71, 91, 111, 131, 151, 171, 191, 211, 251, 271, 291]

for k in k_val:
    classifier = neighbors.KNeighborsClassifier(k, metric='euclidean')
    accuracy = cross_val_score(classifier, x_features, y_encoded, cv=10,␣
 ↪scoring='accuracy').mean()
    print('K =', k, ', Accuracy: ',accuracy)
```

```
K = 3 , Accuracy:  0.78889
K = 5 , Accuracy:  0.7859100000000001
K = 11 , Accuracy:  0.7809999999999999
K = 31 , Accuracy:  0.77349
K = 51 , Accuracy:  0.77732
K = 71 , Accuracy:  0.7784200000000001
K = 91 , Accuracy:  0.7798
K = 111 , Accuracy:  0.77699
K = 131 , Accuracy:  0.77518
K = 151 , Accuracy:  0.76793
K = 171 , Accuracy:  0.76773
K = 191 , Accuracy:  0.7670699999999999
K = 211 , Accuracy:  0.7632599999999999
K = 251 , Accuracy:  0.75099
K = 271 , Accuracy:  0.7526400000000001
K = 291 , Accuracy:  0.7588999999999999
```

[52]:
```
# 10-Fold cross validation on the dataset using manhattan distance
k_val = [3, 5, 11, 31, 51, 71, 91, 111, 131, 151, 171, 191, 211, 251, 271, 291]

for k in k_val:
    classifier = neighbors.KNeighborsClassifier(k, metric='manhattan')
    accuracy = cross_val_score(classifier, x_features, y_encoded, cv=10,␣
 ↪scoring='accuracy').mean()
    print('K =', k, ', Accuracy: ',accuracy)
```

```
K = 3 , Accuracy:  0.8068500000000001
K = 5 , Accuracy:  0.80443
K = 11 , Accuracy:  0.7999099999999999
K = 31 , Accuracy:  0.80442
K = 51 , Accuracy:  0.8047599999999999
K = 71 , Accuracy:  0.80351
K = 91 , Accuracy:  0.80593
K = 111 , Accuracy:  0.80916
K = 131 , Accuracy:  0.8032
K = 151 , Accuracy:  0.8051999999999999
K = 171 , Accuracy:  0.80586
K = 191 , Accuracy:  0.80528
K = 211 , Accuracy:  0.79811
```

```
K = 251 , Accuracy:  0.79233
K = 271 , Accuracy:  0.7952
K = 291 , Accuracy:  0.79949
```

[53]:
```python
from sklearn.model_selection import cross_val_score
from sklearn.metrics import make_scorer, precision_score, recall_score
from sklearn import neighbors

# Assuming x_features and y_encoded are your feature and target variables

k_val = [3, 5, 11, 31, 51, 71, 91, 111, 131, 151, 171, 191, 211, 251, 271, 291]

for k in k_val:
    classifier = neighbors.KNeighborsClassifier(k, metric='euclidean')

    # Calculate accuracy
    accuracy = cross_val_score(classifier, x_features, y_encoded, cv=10,␣
 ↪scoring='accuracy').mean()

    # Calculate precision
    precision = cross_val_score(classifier, x_features, y_encoded, cv=10,␣
 ↪scoring=make_scorer(precision_score)).mean()

    # Calculate recall
    recall = cross_val_score(classifier, x_features, y_encoded, cv=10,␣
 ↪scoring=make_scorer(recall_score)).mean()

    print('K =', k, ', Accuracy:', accuracy, ', Precision:', precision, ',␣
 ↪Recall:', recall)
```

```
K = 3 , Accuracy: 0.78889 , Precision: 0.7919387867242302 , Recall:
0.8069599999999999
K = 5 , Accuracy: 0.7859100000000001 , Precision: 0.7923652881380647 , Recall:
0.7990200000000001
K = 11 , Accuracy: 0.7809999999999999 , Precision: 0.7909925332463366 , Recall:
0.7882199999999999
K = 31 , Accuracy: 0.77349 , Precision: 0.7783442591584715 , Recall:
0.7860199999999999
K = 51 , Accuracy: 0.77732 , Precision: 0.789142914527225 , Recall: 0.7863
K = 71 , Accuracy: 0.7784200000000001 , Precision: 0.7934212935475051 , Recall:
0.7827399999999999
K = 91 , Accuracy: 0.7798 , Precision: 0.7995343447953918 , Recall:
0.7776599999999999
K = 111 , Accuracy: 0.77699 , Precision: 0.7980708668900063 , Recall: 0.77672
K = 131 , Accuracy: 0.77518 , Precision: 0.7951990342762876 , Recall:
0.7774599999999999
K = 151 , Accuracy: 0.76793 , Precision: 0.7840193019598036 , Recall: 0.7761
```

```
K = 171 , Accuracy: 0.76773 , Precision: 0.7834847144040152 , Recall:
0.7787200000000001
K = 191 , Accuracy: 0.7670699999999999 , Precision: 0.7818282331169801 , Recall:
0.7796799999999999
K = 211 , Accuracy: 0.7632599999999999 , Precision: 0.7766213825172243 , Recall:
0.78124
K = 251 , Accuracy: 0.75099 , Precision: 0.7628746806354341 , Recall: 0.77332
K = 271 , Accuracy: 0.7526400000000001 , Precision: 0.7633079061165955 , Recall:
0.7741
K = 291 , Accuracy: 0.7588999999999999 , Precision: 0.767609312272534 , Recall:
0.78248
```

[54]:
```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import make_scorer, precision_score, recall_score
from sklearn.neighbors import KNeighborsClassifier

# Assuming x_features and y_encoded are your feature and target variables

k_val = [3, 5, 11, 31, 51, 71, 91, 111, 131, 151, 171, 191, 211, 251, 271, 291]

for k in k_val:
    classifier = KNeighborsClassifier(k, metric='manhattan',weights='distance')
#Hyper-Parameters changed - Distance
    # Calculate accuracy
    accuracy = cross_val_score(classifier, x_features, y_encoded, cv=10,␣
 ↪scoring='accuracy').mean()

    # Calculate precision
    precision = cross_val_score(classifier, x_features, y_encoded, cv=10,␣
 ↪scoring=make_scorer(precision_score)).mean()

    # Calculate recall
    recall = cross_val_score(classifier, x_features, y_encoded, cv=10,␣
 ↪scoring=make_scorer(recall_score)).mean()

    print('K =', k, ', Accuracy:', accuracy, ', Precision:', precision, ',␣
 ↪Recall:', recall)
```

```
K = 3 , Accuracy: 0.8068500000000001 , Precision: 0.8108903070068443 , Recall:
0.81932
K = 5 , Accuracy: 0.80468 , Precision: 0.8098712379509271 , Recall:
0.8141400000000001
K = 11 , Accuracy: 0.8002499999999999 , Precision: 0.808418707712533 , Recall:
0.80282
K = 31 , Accuracy: 0.80493 , Precision: 0.813872612997249 , Recall: 0.80754
K = 51 , Accuracy: 0.80578 , Precision: 0.8148582210695465 , Recall: 0.81182
K = 71 , Accuracy: 0.8049000000000002 , Precision: 0.811230972699241 , Recall:
```

```
0.81368
K = 91 , Accuracy: 0.80758 , Precision: 0.8144153797021488 , Recall:
0.8168399999999998
K = 111 , Accuracy: 0.8103 , Precision: 0.8170783864180245 , Recall: 0.82358
K = 131 , Accuracy: 0.8063499999999999 , Precision: 0.8119991271666948 , Recall:
0.8263999999999999
K = 151 , Accuracy: 0.8059200000000001 , Precision: 0.8100704597521051 , Recall:
0.8303
K = 171 , Accuracy: 0.8091900000000001 , Precision: 0.8111625696217388 , Recall:
0.83754
K = 191 , Accuracy: 0.8075800000000001 , Precision: 0.8082519918995864 , Recall:
0.83894
K = 211 , Accuracy: 0.80467 , Precision: 0.8045483698381322 , Recall:
0.8396000000000001
K = 251 , Accuracy: 0.79833 , Precision: 0.7947938093620234 , Recall:
0.8435599999999999
K = 271 , Accuracy: 0.8002100000000001 , Precision: 0.7958020910329648 , Recall:
0.8468000000000002
K = 291 , Accuracy: 0.8025399999999999 , Precision: 0.7959285629673175 , Recall:
0.8522000000000001
```

[ ]: