KNN - K Nearest Neighbors - Classification

To understand KNN for classification, we'll work with a simple dataset representing gene expression levels. Gene expression levels are calculated by the ratio between the expression of the target gene (i.e., the gene of interest) and the expression of one or more reference genes (often household genes). This dataset is synthetic and specifically designed to show some of the strengths and limitations of using KNN for Classification.

More info on gene expression: https://www.sciencedirect.com/topics/biochemistry-genetics-and-molecular-biology/gene-expression-level

Imports

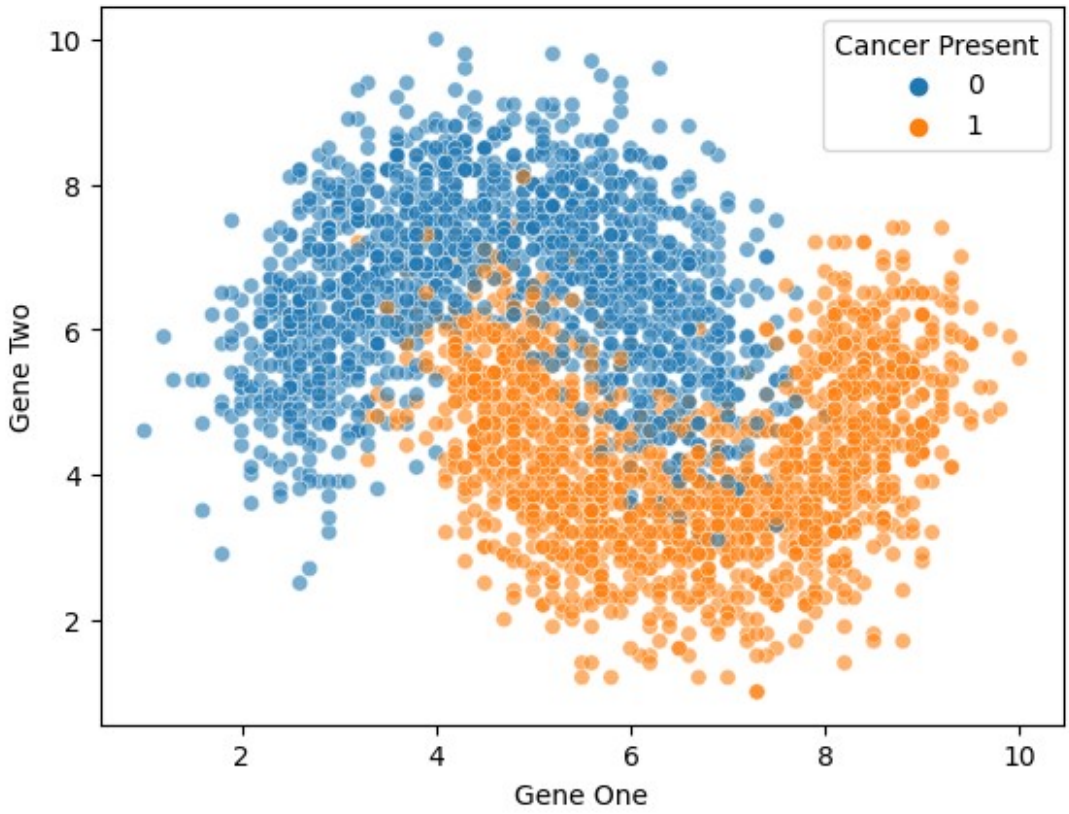
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
df= pd.read_csv("D:\\Study\Programming\\python\\Python course from udemy\\Udemy - 2022 Python for Machine Learning & Data Science Masterclass\\01 - Introduction to Course\\1UNZIP-FOR-NOTEBOOKS-FINAL\\DATA\\gene_expression.csv")
df.head()
```

	Gene One	Gene Two	Cancer Present
0	4.3	3.9	1
1	2.5	6.3	0
2	5.7	3.9	1
3	6.1	6.2	0
4	7.4	3.4	1

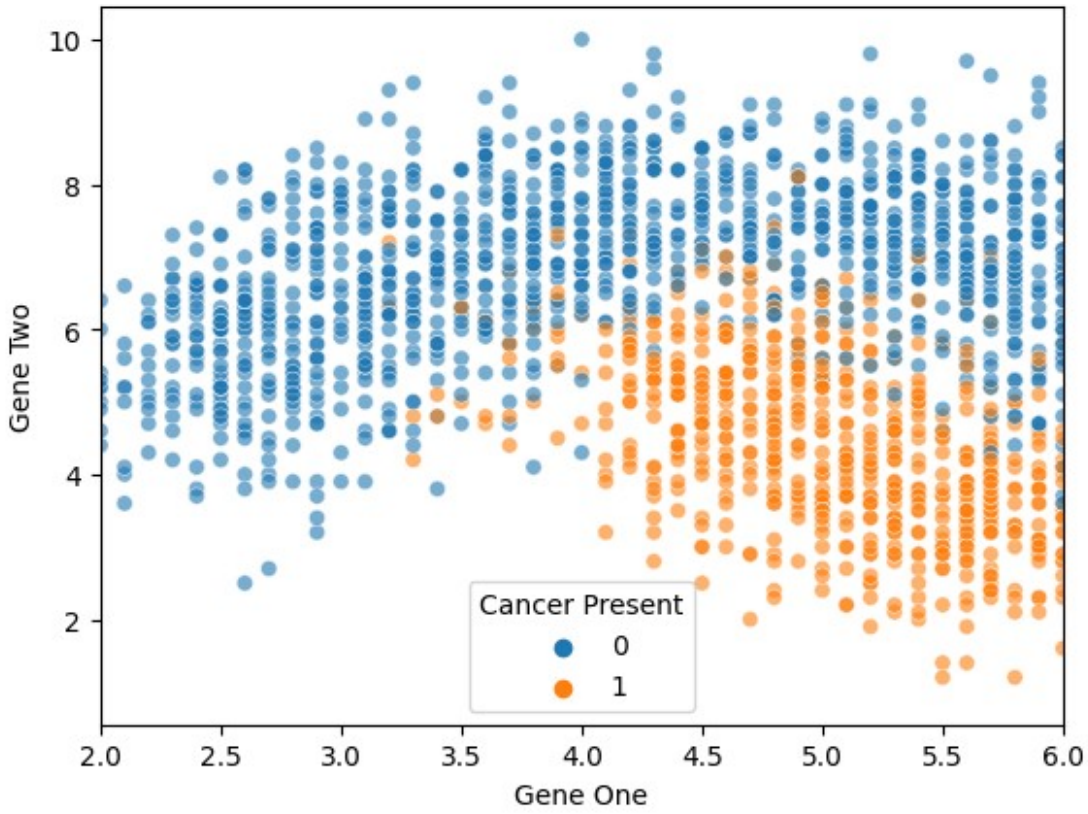
```
sns.scatterplot(x='Gene One',y='Gene Two',hue='Cancer Present',data=df,alpha=0.6)
```

```
<AxesSubplot: xlabel='Gene One', ylabel='Gene Two'>
```



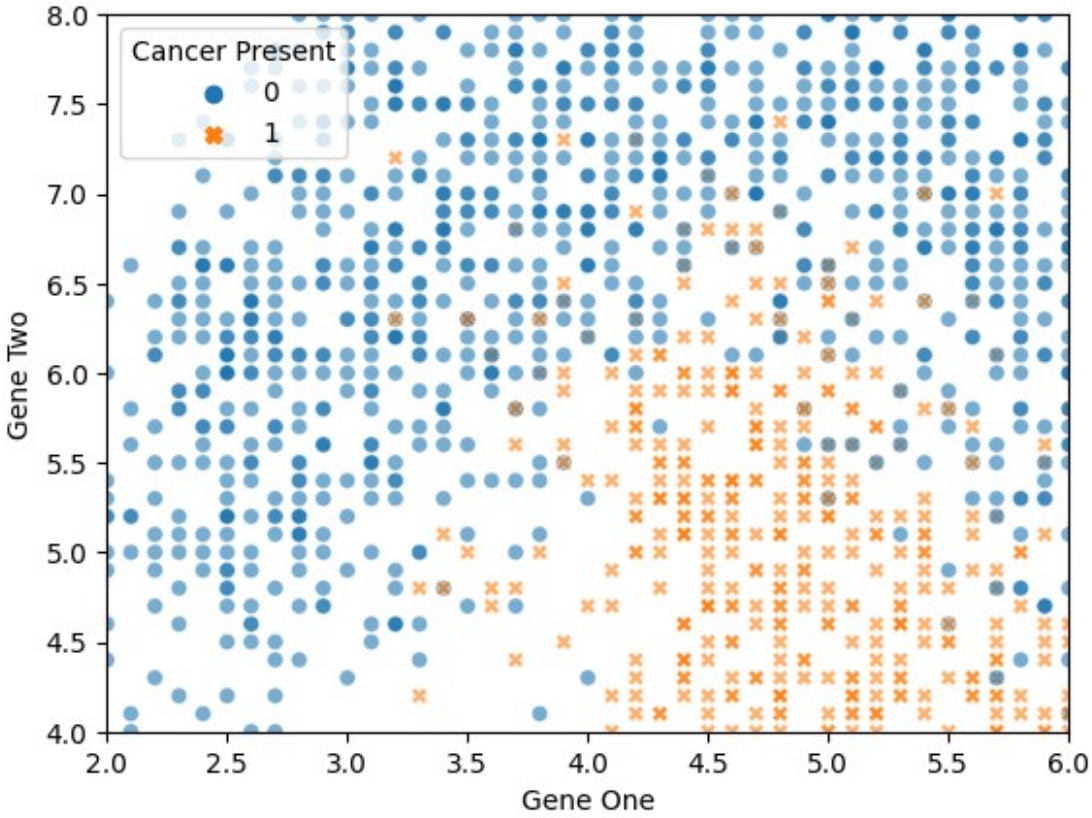

```
# Here we set limit on x-axis to zoom
sns.scatterplot(x='Gene One',y='Gene Two',hue='Cancer
Present',data=df,alpha=0.6)

plt.xlim(2,6)
```

```
(2.0, 6.0)
```



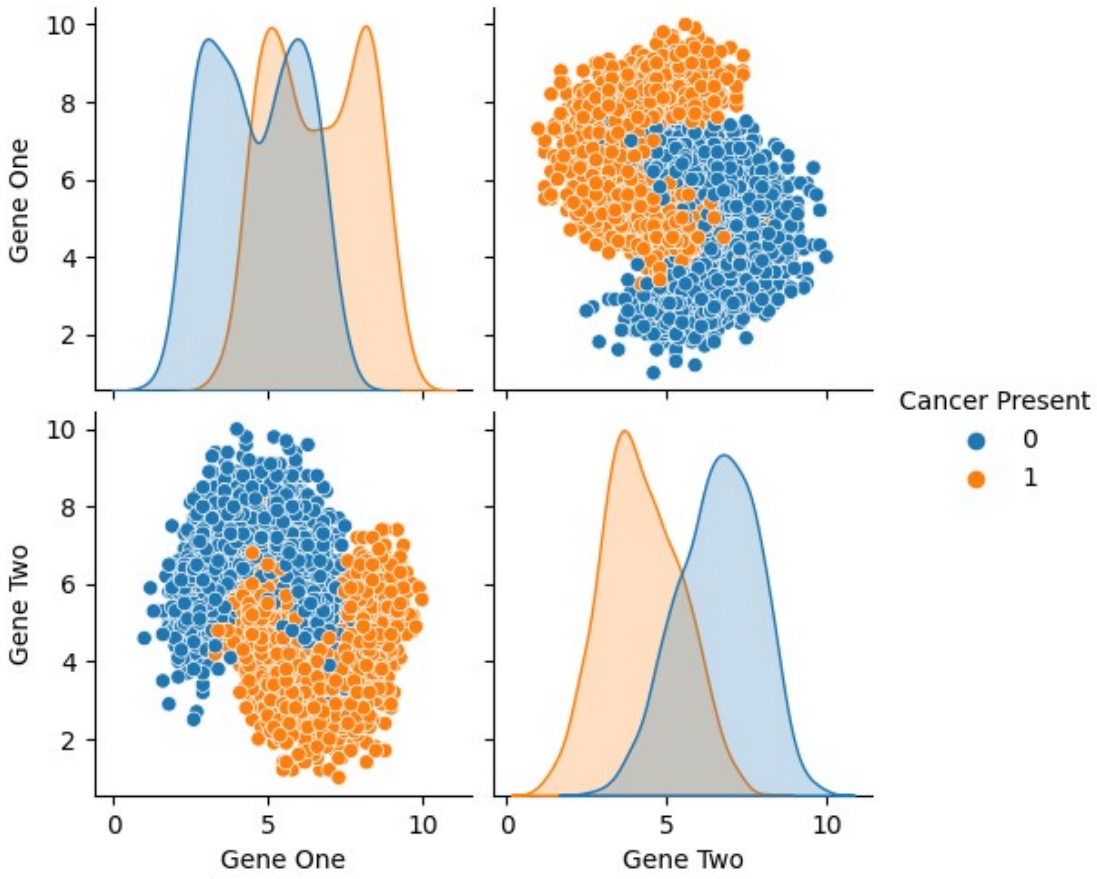

```
# Here zoomed more
sns.scatterplot(x='Gene One',y='Gene Two',hue='Cancer
Present',data=df,alpha=0.6,style='Cancer Present')

plt.xlim(2,6)
plt.ylim(4,8)
```

```
(4.0, 8.0)
```




```
sns.pairplot(data=df,hue='Cancer Present')
```

```
<seaborn.axisgrid.PairGrid at 0x14d081154f0>
```




```
X=df.drop('Cancer Present',axis=1)
y=df['Cancer Present']
```

Train|Test Split and Scaling Data

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,rando
m_state=42)

scaler=StandardScaler()

scaler_x_train=scaler.fit_transform(X_train)
scaler_x_test=scaler.fit_transform(X_test)

from sklearn.neighbors import KNeighborsClassifier

#help(KNeighborsClassifier)

knn_model=KNeighborsClassifier(n_neighbors=1)

knn_model.fit(scaler_x_train,y_train)

KNeighborsClassifier(n_neighbors=1)
```

```
y_pred = knn_model.predict(scaler_x_test)
```

Model Evaluation

```
from sklearn.metrics import
confusion_matrix,classification_report,accuracy_score

confusion_matrix(y_test,y_pred)
```

```
array([[426,  44],
       [ 34, 396]], dtype=int64)
```

```
len(y_test)
```

```
900
```

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.93      0.91      0.92       470
           1       0.90      0.92      0.91       430

    accuracy                           0.91       900
   macro avg       0.91      0.91      0.91       900
weighted avg       0.91      0.91      0.91       900
```

```
df['Cancer Present'].value_counts()
```

```
1    1500
0    1500
Name: Cancer Present, dtype: int64
```

```
accuracy_score(y_test,y_pred)
```

```
0.9133333333333333
```

```
error = 1 - accuracy_score(y_test,y_pred)
error
```

```
0.08666666666666667
```

Elbow Method for Choosing Reasonable K Values

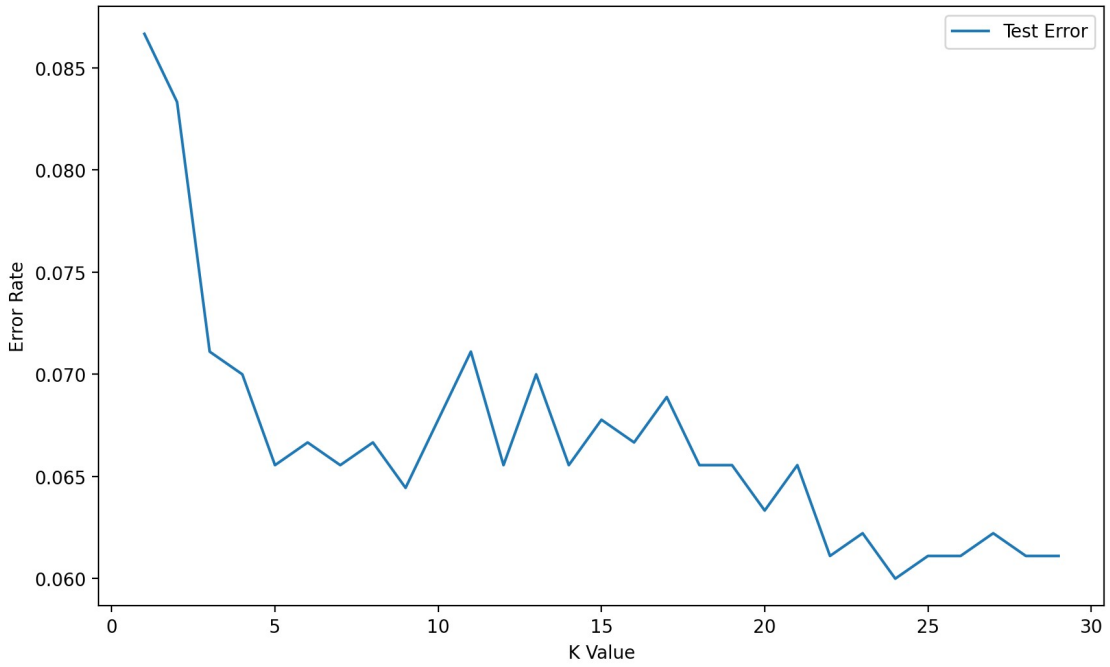
NOTE: This uses the test set for the hyperparameter selection of K.

```
test_error_rates = []

for k in range(1,30):
    knn_model=KNeighborsClassifier(n_neighbors=k)
    knn_model.fit(scaler_x_train,y_train)

    y_pred_test=knn_model.predict(scaler_x_test)
```

```
    test_error = 1 - accuracy_score(y_pred_test,y_test)
    test_error_rates.append(test_error)
```

```
plt.figure(figsize=(10,6),dpi=200)
plt.plot(range(1,30),test_error_rates,label='Test Error')
plt.legend()
plt.ylabel('Error Rate')
plt.xlabel('K Value')
```

```
Text(0.5, 0, 'K Value')
```



Full Cross Validation Grid Search for K Value

Creating a Pipeline to find K value

Follow along very carefully here! We use very specific string codes AND variable names here so that everything matches up correctly. This is not a case where you can easily swap out variable names for whatever you want!

We'll use a Pipeline object to set up a workflow of operations:

1. Scale Data
2. Create Model on Scaled Data

---

*How does the Scaler work inside a Pipeline with CV? Is scikit-learn "smart" enough to understand .fit() on train vs .transform() on train and test?**

Yes! Scikit-Learn's pipeline is well suited for this! Full Info in Documentation

When you use the StandardScaler as a step inside a Pipeline then scikit-learn will internally do the job for you.

What happens can be discribed as follows:

- Step 0: The data are split into TRAINING data and TEST data according to the cv parameter that you specified in the GridSearchCV.
- Step 1: the scaler is fitted on the TRAINING data
- Step 2: the scaler transforms TRAINING data
- Step 3: the models are fitted/trained using the transformed TRAINING data
- Step 4: the scaler is used to transform the TEST data
- Step 5: the trained models predict using the transformed TEST data

---

```
scaler = StandardScaler()
knn = KNeighborsClassifier()

# Here we see all the keys values of KNeighborsClassifier
knn.get_params().keys()

dict_keys(['algorithm', 'leaf_size', 'metric', 'metric_params',
'n_jobs', 'n_neighbors', 'p', 'weights'])

# Highly recommend string code matches variable name!
operations = [('scaler',scaler),('knn',knn)]

from sklearn.pipeline import Pipeline

pipe = Pipeline(operations)

from sklearn.model_selection import GridSearchCV
```

---

*Note: If your parameter grid is going inside a PipeLine, your parameter name needs to be specified in the following manner:**

- chosen_string_name + **two** underscores + parameter key name
- model_name + __ + parameter name
- knn_model + __ + n_neighbors
- knn_model__n_neighbors

StackOverflow on this

The reason we have to do this is because it let's scikit-learn know what operation in the pipeline these parameters are related to (otherwise it might think n_neighbors was a parameter in the scaler).

```
k_values = list(range(1,20))
k_values
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

```
# check that latest markdown cell and we will find out how to write
this code
param_grid = {'knn__n_neighbors': k_values}

full_cv_classifier =
GridSearchCV(pipe,param_grid,cv=5,scoring='accuracy')

full_cv_classifier.fit(X_train,y_train)
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('scaler', StandardScaler()),
                                       ('knn',
KNeighborsClassifier())]),
             param_grid={'knn__n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8,
9, 10, 11,
                                              12, 13, 14, 15, 16, 17,
18, 19]},
             scoring='accuracy')

full_cv_classifier.best_estimator_.get_params()
```

```
{'memory': None,
 'steps': [('scaler', StandardScaler()),
  ('knn', KNeighborsClassifier(n_neighbors=16))],
 'verbose': False,
 'scaler': StandardScaler(),
 'knn': KNeighborsClassifier(n_neighbors=16),
 'scaler__copy': True,
 'scaler__with_mean': True,
 'scaler__with_std': True,
 'knn__algorithm': 'auto',
 'knn__leaf_size': 30,
 'knn__metric': 'minkowski',
 'knn__metric_params': None,
 'knn__n_jobs': None,
 'knn__n_neighbors': 16,
 'knn__p': 2,
 'knn__weights': 'uniform'}

full_cv_classifier.cv_results_.keys()
```

```
dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time',
'std_score_time', 'param_knn__n_neighbors', 'params',
'split0_test_score', 'split1_test_score', 'split2_test_score',
'split3_test_score', 'split4_test_score', 'mean_test_score',
'std_test_score', 'rank_test_score'])
```

```
full_pred = full_cv_classifier.predict(X_test)

print(classification_report(full_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       0.96      0.94      0.95       480
           1       0.93      0.95      0.94       420

    accuracy                           0.94       900
   macro avg       0.94      0.94      0.94       900
weighted avg       0.94      0.94      0.94       900
```

```
new_patient = [[3.8,6.4]]

# From the result we can see that  is 0 means that person doesnt have
cancer
full_cv_classifier.predict(new_patient)
```

```
C:\Users\Chromsy\AppData\Roaming\Python\Python39\site-packages\
sklearn\base.py:420: UserWarning: X does not have valid feature names,
but StandardScaler was fitted with feature names
  warnings.warn(

array([0], dtype=int64)
```

```
# Here we see the percentage of cancer , 1 means 100% for 0 and 0 mean
0% for 1
full_cv_classifier.predict_proba(new_patient)
```

```
C:\Users\Chromsy\AppData\Roaming\Python\Python39\site-packages\
sklearn\base.py:420: UserWarning: X does not have valid feature names,
but StandardScaler was fitted with feature names
  warnings.warn(

array([[1., 0.]])
```