[(http://www.pieriandata.com)](http://www.pieriandata.com)

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 (https://www.sciencedirect.com/topics/biochemistry-genetics-and-molecular-biology/gene-expression-level)](https://www.sciencedirect.com/topics/biochemistry-genetics-and-molecular-biology/gene-expression-level)

Imports

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Data

In [2]:
```
df = pd.read_csv('gene_expression.csv')
```

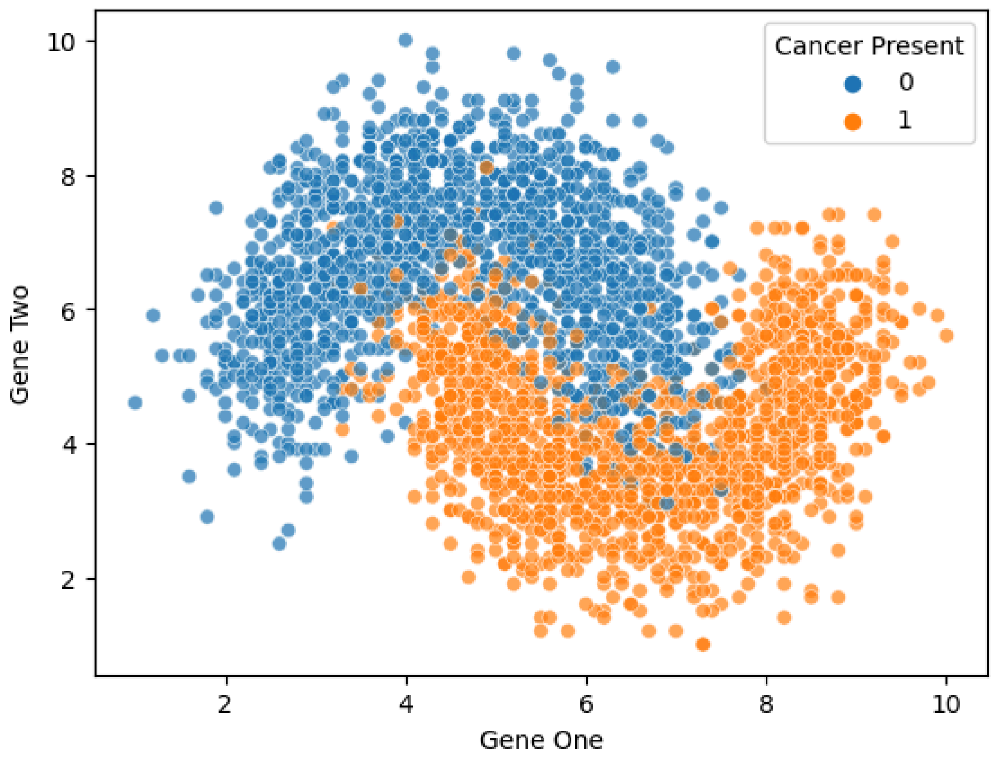
In [3]:
```
df.head()
```

Out[3]:

	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

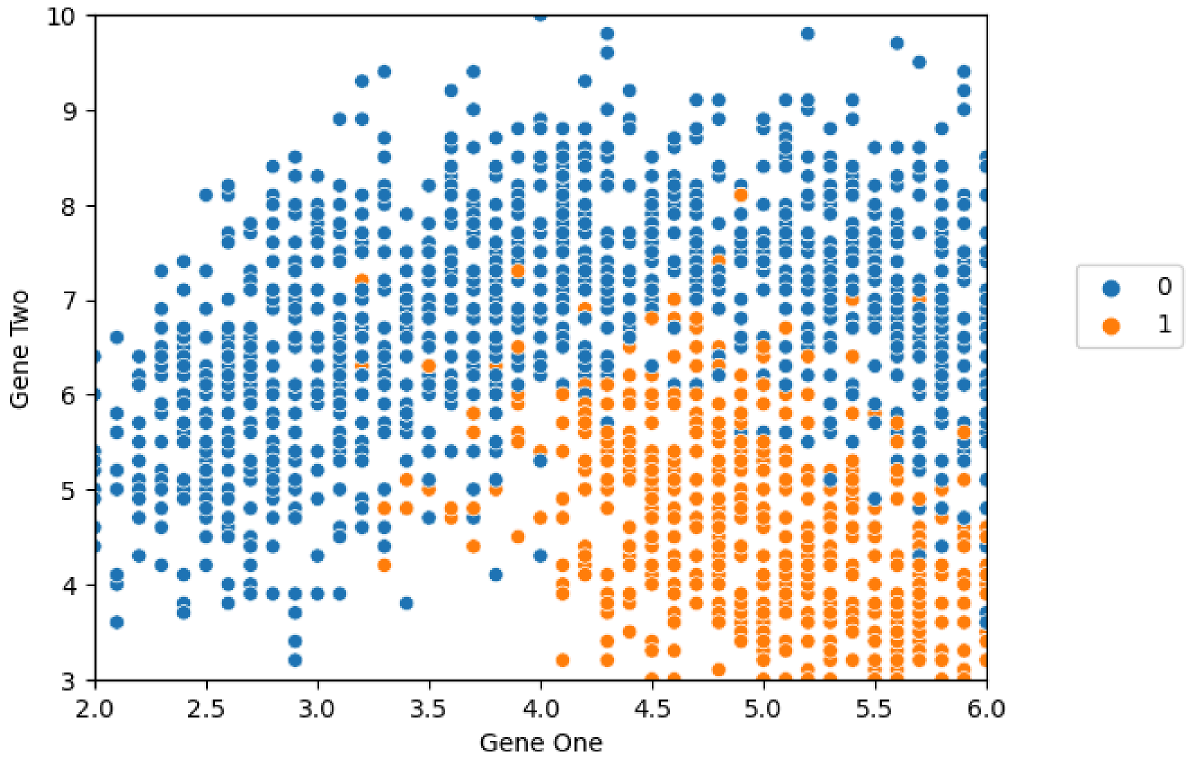
```
In [4]: sns.scatterplot(x='Gene One',y='Gene Two',hue='Cancer Present',data=df,alpha=0.7)
```

```
Out[4]: <AxesSubplot:xlabel='Gene One', ylabel='Gene Two'>
```



```
In [5]: sns.scatterplot(x='Gene One',y='Gene Two',hue='Cancer Present',data=df)
        plt.xlim(2,6)
        plt.ylim(3,10)
        plt.legend(loc=(1.1,0.5))
```

```
Out[5]: <matplotlib.legend.Legend at 0x21a3a9253a0>
```



Train|Test Split and Scaling Data

In [6]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [7]:
```
X = df.drop('Cancer Present',axis=1)
y = df['Cancer Present']
```

In [8]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
```

In [9]:
```
scaler = StandardScaler()
```

In [10]:
```
scaled_X_train = scaler.fit_transform(X_train)
scaled_X_test = scaler.transform(X_test)
```

In [11]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [12]:
```
knn_model = KNeighborsClassifier(n_neighbors=1)
```

In [13]:
```
knn_model.fit(scaled_X_train,y_train)
```

Out[13]:
```
KNeighborsClassifier(n_neighbors=1)
```

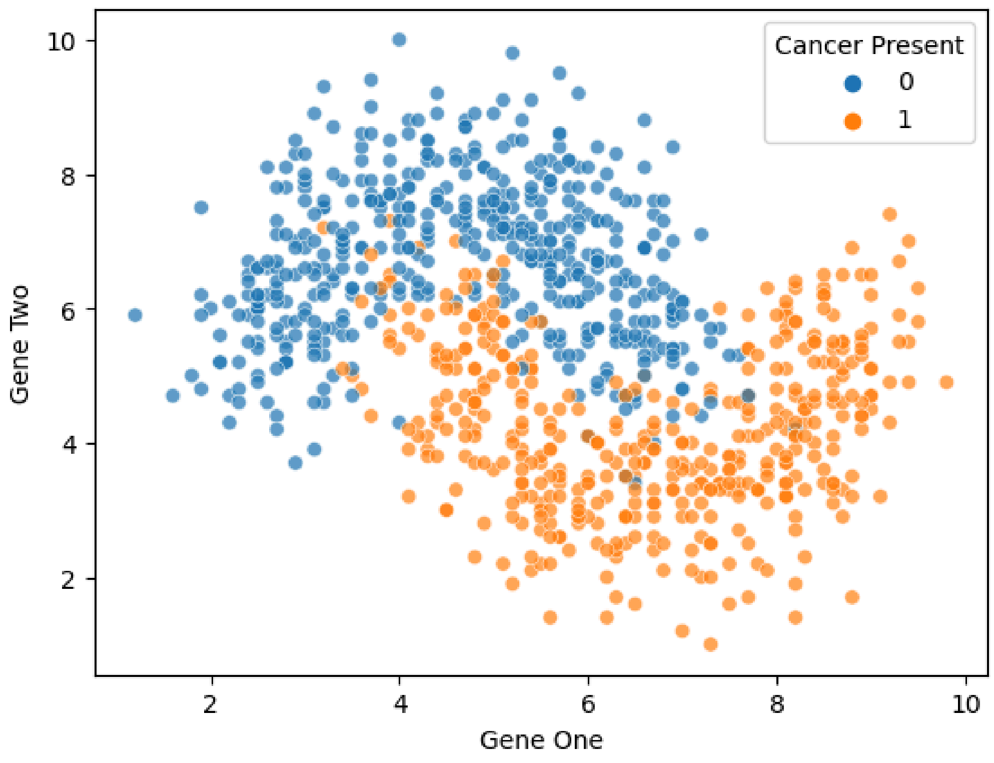
Understanding KNN and Choosing K Value

In [14]:
```
full_test = pd.concat([X_test,y_test],axis=1)
```

In [15]:
```
len(full_test)
```

Out[15]:
```
900
```

```
In [16]: sns.scatterplot(x='Gene One',y='Gene Two',hue='Cancer Present',
                         data=full_test,alpha=0.7)

Out[16]: <AxesSubplot:xlabel='Gene One', ylabel='Gene Two'>
```



Model Evaluation

```
In [18]: import warnings
         warnings.filterwarnings('ignore')

         y_pred = knn_model.predict(scaled_X_test)

In [19]: from sklearn.metrics import classification_report,confusion_matrix,accuracy_score

In [20]: accuracy_score(y_test,y_pred)

Out[20]: 0.9011111111111111

In [21]: confusion_matrix(y_test,y_pred)

Out[21]: array([[426,  44],
                [ 45, 385]], dtype=int64)

In [22]: print(classification_report(y_test,y_pred))

                       precision    recall  f1-score   support

                    0       0.90      0.91      0.91       470
                    1       0.90      0.90      0.90       430

             accuracy                           0.90       900
            macro avg       0.90      0.90      0.90       900
         weighted avg       0.90      0.90      0.90       900
```

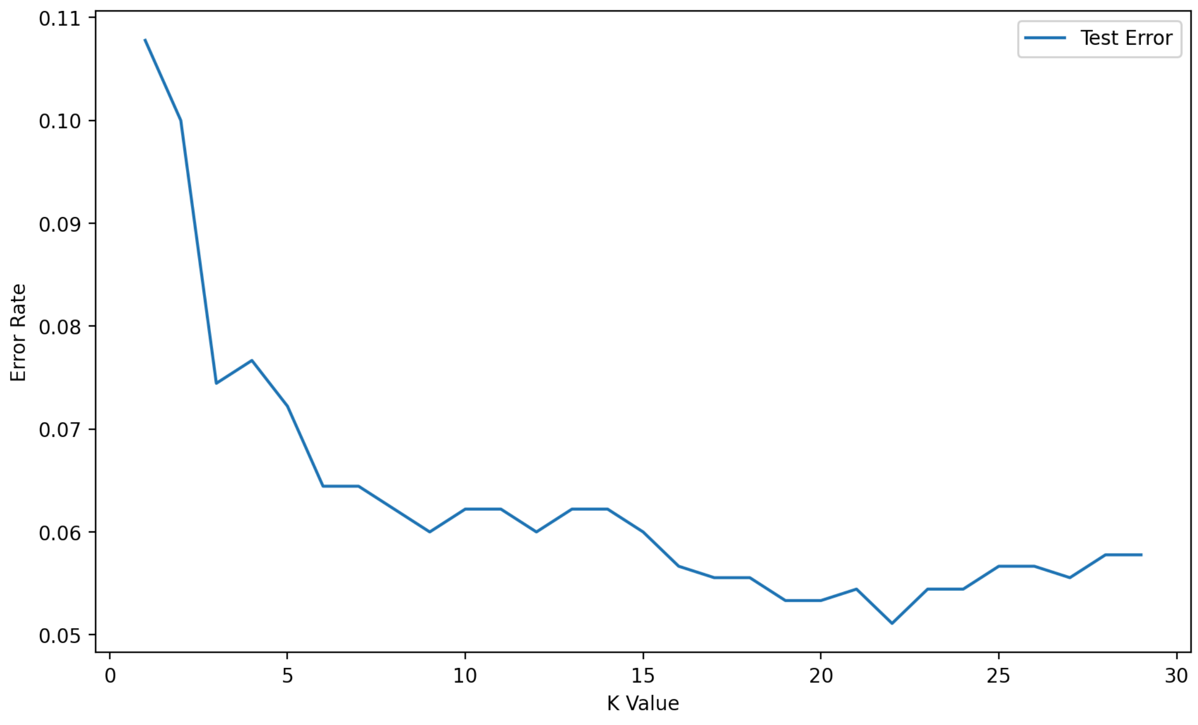
Elbow Method for Choosing Reasonable K Values

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

In [23]:
```
test_error_rates = []


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

    y_pred_test = knn_model.predict(scaled_X_test)

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

In [24]:
```
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")
```

Out[24]:
```
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 (https://scikit-learn.org/stable/modules/preprocessing.html#standardization-or-mean-removal-and-variance-scaling)](https://scikit-learn.org/stable/modules/preprocessing.html#standardization-or-mean-removal-and-variance-scaling) **

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

---

In [24]:
```
scaler = StandardScaler()
```

In [25]:
```
knn = KNeighborsClassifier()
```

In [26]:
```
knn.get_params().keys()
```

Out[26]:
```
dict_keys(['algorithm', 'leaf_size', 'metric', 'metric_params', 'n_jobs', 'n_neighbor
s', 'p', 'weights'])
```

In [27]:
```
# Highly recommend string code matches variable name!
operations = [('scaler',scaler),('knn',knn)]
```

In [28]:
```
from sklearn.pipeline import Pipeline
```

In [30]:
```
pipe = Pipeline(operations)
```

In [31]:
```
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 (https://stackoverflow.com/questions/41899132/invalid-parameter-for-sklearn-estimator-pipeline)](https://stackoverflow.com/questions/41899132/invalid-parameter-for-sklearn-estimator-pipeline)

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).

In [32]:
```
k_values = list(range(1,20))
```

In [33]:
```
k_values
```

Out[33]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

In [34]:
```

param_grid = {'knn__n_neighbors': k_values}
```

In [35]:
```
full_cv_classifier = GridSearchCV(pipe,param_grid,cv=5,scoring='accuracy')
```

In [36]:
```
# Use full X and y if you DON'T want a hold-out test set
# Use X_train and y_train if you DO want a holdout test set (X_test,y_test)
full_cv_classifier.fit(X_train,y_train)
```

Out[36]:
```
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')
```

In [36]:
```
full_cv_classifier.best_estimator_.get_params()
```

Out[36]:
```
{'memory': None,
 'steps': [('scaler', StandardScaler()),
  ('knn', KNeighborsClassifier(n_neighbors=14))],
 'verbose': False,
 'scaler': StandardScaler(),
 'knn': KNeighborsClassifier(n_neighbors=14),
 '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': 14,
 'knn__p': 2,
 'knn__weights': 'uniform'}
```

In [37]:
```
full_cv_classifier.cv_results_.keys()
```

Out[37]:
```
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_sc
ore', 'split3_test_score', 'split4_test_score', 'mean_test_score', 'std_test_score', 'r
ank_test_score'])
```

Let's check our understanding: **How many total runs did we do?**

In [38]:
```
len(k_values)
```

Out[38]:
```
19
```

In [39]:
```
full_cv_classifier.cv_results_['mean_test_score']
```

Out[39]:
```
array([0.90380952, 0.90714286, 0.92142857, 0.91380952, 0.92380952,
       0.92047619, 0.92761905, 0.9252381 , 0.9247619 , 0.92285714,
       0.9252381 , 0.92428571, 0.92761905, 0.92809524, 0.92857143,
       0.93      , 0.92904762, 0.92857143, 0.92761905])
```

In [40]:
```
len(full_cv_classifier.cv_results_['mean_test_score'])
```
Out[40]:
```
19
```

Final Model

We just saw that our GridSearch recommends a K=14 (in line with our alternative Elbow Method). Let's now use the PipeLine again, but this time, no need to do a grid search, instead we will evaluate on our hold-out Test Set.

In [41]:
```
scaler = StandardScaler()
knn14 = KNeighborsClassifier(n_neighbors=14)
operations = [('scaler',scaler),('knn14',knn14)]
```
In [42]:
```
pipe = Pipeline(operations)
```
In [43]:
```
pipe.fit(X_train,y_train)
```
Out[43]:
```
Pipeline(steps=[('scaler', StandardScaler()),
                ('knn14', KNeighborsClassifier(n_neighbors=14))])
```
In [45]:
```
pipe_pred = pipe.predict(X_test)
```
In [46]:
```
print(classification_report(y_test,pipe_pred))
```
```
              precision    recall  f1-score   support

           0       0.93      0.95      0.94       470
           1       0.95      0.92      0.93       430

    accuracy                           0.94       900
   macro avg       0.94      0.94      0.94       900
weighted avg       0.94      0.94      0.94       900
```
In [47]:
```
single_sample = X_test.iloc[40]
```
In [48]:
```
single_sample
```
Out[48]:
```
Gene One    3.8
Gene Two    6.3
Name: 194, dtype: float64
```
In [49]:
```
pipe.predict(single_sample.values.reshape(1, -1))
```
Out[49]:
```
array([0], dtype=int64)
```
In [50]:
```
pipe.predict_proba(single_sample.values.reshape(1, -1))
```
Out[50]:
```
array([[0.92857143, 0.07142857]])
```