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.

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

Import Data

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

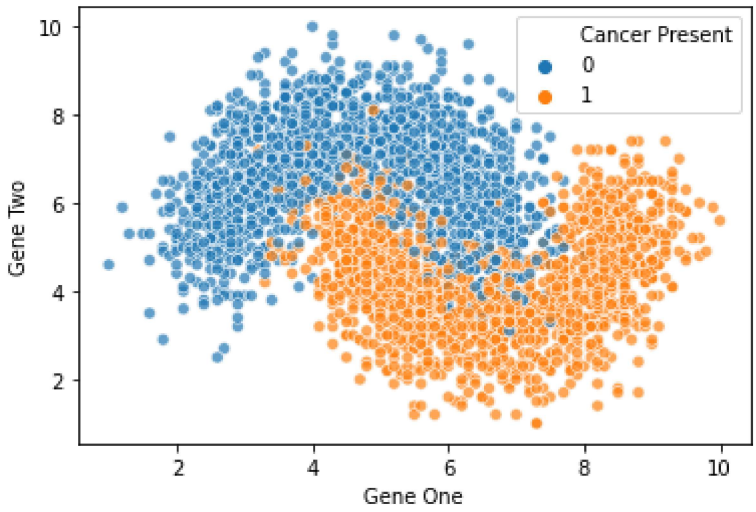
Out[2]:

	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

Exploratory Data Analysis

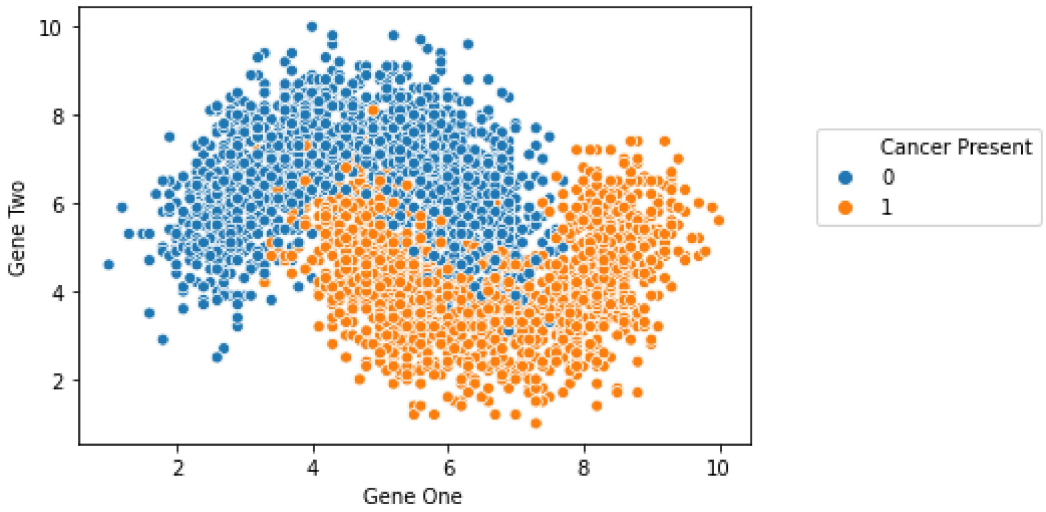
In [3]:
```
sns.scatterplot(x='Gene One',y='Gene Two',hue='Cancer Present',data=df,alpha=
0.7)
```

Out[3]: <matplotlib.axes._subplots.AxesSubplot at 0x21ccea31880>



In [4]:
```
sns.scatterplot(x='Gene One',y='Gene Two',hue='Cancer Present',data=df)
plt.legend(loc=(1.1,0.5))
```

Out[4]: <matplotlib.legend.Legend at 0x21ccf1bd730>



X & y

In [5]:
```
X = df.drop('Cancer Present',axis=1)
y = df['Cancer Present']
```

Train|Test Split

In [6]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rando
m_state=42)
```

Scaling Data

In [7]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Modelling

In [8]:
```
from sklearn.neighbors import KNeighborsClassifier

knn_model = KNeighborsClassifier(n_neighbors=1)

knn_model.fit(X_train,y_train)
```

Out[8]:
```
KNeighborsClassifier(n_neighbors=1)
```

Model Evaluation

In [9]:
```
y_pred = knn_model.predict(X_test)
```

In [10]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```

Out[10]:
```
0.8922222222222222
```

In [11]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,y_pred)
```

Out[11]:
```
array([[420,  50],
       [ 47, 383]], dtype=int64)
```

In [12]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.90      0.89      0.90       470
           1       0.88      0.89      0.89       430

    accuracy                           0.89       900
   macro avg       0.89      0.89      0.89       900
weighted avg       0.89      0.89      0.89       900
```

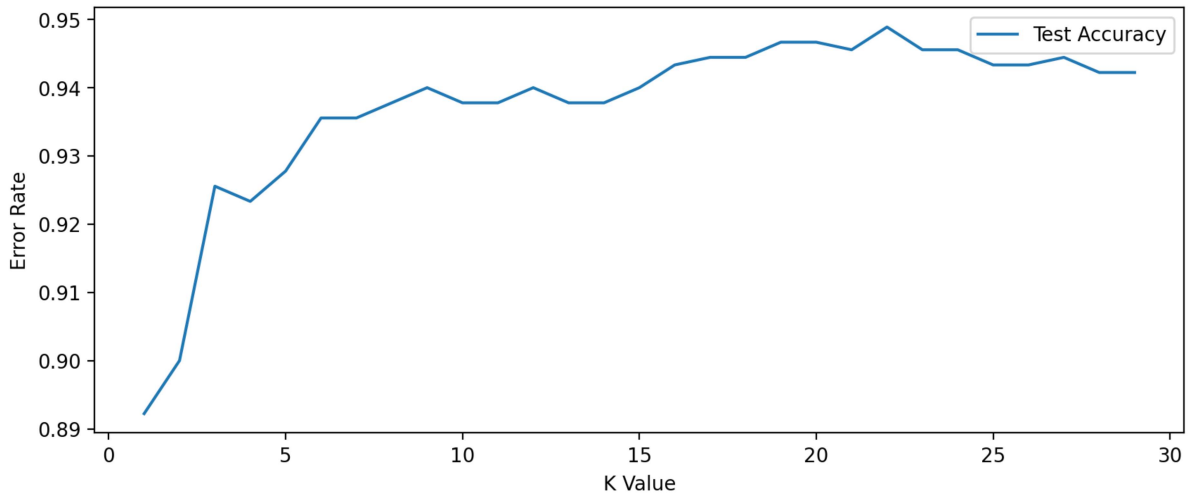
Elbow Method for Choosing Reasonable K Values

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

In [13]:
```
test_acc = []

for k in range(1,30):
    knn_model = KNeighborsClassifier(n_neighbors=k)
    knn_model.fit(X_train,y_train)
    y_pred_test = knn_model.predict(X_test)
    test_acc_score = accuracy_score(y_test,y_pred_test)

    test_acc.append(test_acc_score)

test_acc
```

Out[13]:
```
[0.8922222222222222,
 0.9,
 0.9255555555555556,
 0.9233333333333333,
 0.9277777777777778,
 0.9355555555555556,
 0.9355555555555556,
 0.9377777777777778,
 0.94,
 0.9377777777777778,
 0.9377777777777778,
 0.94,
 0.9377777777777778,
 0.9377777777777778,
 0.94,
 0.9433333333333334,
 0.9444444444444444,
 0.9444444444444444,
 0.9466666666666667,
 0.9466666666666667,
 0.9455555555555556,
 0.9488888888888889,
 0.9455555555555556,
 0.9455555555555556,
 0.9433333333333334,
 0.9433333333333334,
 0.9444444444444444,
 0.9422222222222222,
 0.9422222222222222]
```

In [14]:
```
plt.figure(figsize=(10,4),dpi=200)
plt.plot(range(1,30),test_acc,label='Test Accuracy')

plt.ylabel('Error Rate')
plt.xlabel("K Value")

plt.legend()
plt.show()
```



Understanding KNN and Choosing K Value

Full Cross Validation Grid Search for K Value

In [15]:
```
param_grid = {'n_neighbors': list(range(1,30))}
```

In [16]:
```
from sklearn.model_selection import GridSearchCV
full_cv_classifier = GridSearchCV(knn_model,param_grid,cv=5,scoring='accuracy'
)
full_cv_classifier.fit(X_train,y_train)
```

Out[16]:
```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(n_neighbors=29),
             param_grid={'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 1
2,
                                         13, 14, 15, 16, 17, 18, 19, 20, 21,
22,
                                         23, 24, 25, 26, 27, 28, 29]},
             scoring='accuracy')
```

In [17]:
```
full_cv_classifier.best_params_
```

Out[17]:
```
{'n_neighbors': 20}
```

In [18]:
```
full_cv_classifier.cv_results_['mean_test_score']
```

Out[18]:
```
array([0.90142857, 0.90333333, 0.91761905, 0.91190476, 0.92333333,
       0.92238095, 0.9252381 , 0.92428571, 0.9252381 , 0.92142857,
       0.92428571, 0.92428571, 0.92809524, 0.92904762, 0.92761905,
       0.92761905, 0.92809524, 0.92904762, 0.92714286, 0.93095238,
       0.92904762, 0.93095238, 0.92809524, 0.92857143, 0.92761905,
       0.92666667, 0.92857143, 0.92904762, 0.92904762])
```

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 [19]:
```
knn = KNeighborsClassifier(n_neighbors=20)

knn.fit(X_train,y_train)

y_pred = knn.predict(X_test)
```

In [20]:
```
accuracy_score(y_test,y_pred)
```

Out[20]:
```
0.9466666666666667
```

In [21]:
```
confusion_matrix(y_test,y_pred)
```

Out[21]:
```
array([[451,  19],
       [ 29, 401]], dtype=int64)
```

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

```
              precision    recall  f1-score   support

           0       0.94      0.96      0.95       470
           1       0.95      0.93      0.94       430

    accuracy                           0.95       900
   macro avg       0.95      0.95      0.95       900
weighted avg       0.95      0.95      0.95       900
```

In [23]:
```
#Training Accuracy
y_pred_train = knn.predict(X_train)
accuracy_score(y_train,y_pred_train)
```

Out[23]:
```
0.9333333333333333
```