Naive Bayes

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is good to know on this algorithm, each feature is considered independent of other features. Second thing we should notice is each feature is given to the same weight. This algorithm might not perform good in real world situation. However it is good for practice.

Bayes equation

Probability of B occurring given evidence A has already occurred

Probability of A occurring

$$P(\mathrm{A}|\mathrm{B}) = \frac{P(\mathrm{B}|A) \cdot P(A)}{P(B)}$$

Probability of A occurring given evidence B has already occurred

Probability of B occurring

- P(A|B) also called posterior.
- P(B|A) and how likely A is on its own.

Basically, we are trying to find probability of event A, given the event B is true. Event B is also termed as evidence.

Naive Assumption

with naive assumption, we split all the feaetures into independent features. if 2 events are independent they would follow the equation down below:

```
P(A,B) = P(A)P(B)
```

consider all features of our dataset on array X

$$P(y|x_1, ..., x_n) = \frac{P(x_1|y)P(x_2|y)...P(x_n|y)P(y)}{P(x_1)P(x_2)...P(x_n)}$$

the bayse equation would changed like the following equation.

$$P(y|x_1, ..., x_n) = \frac{P(x_1|y)P(x_2|y)...P(x_n|y)P(y)}{P(x_1)P(x_2)...P(x_n)}$$

Now, you can obtain the values for each by looking at the dataset and substitute them into the equation.

In [43]:
```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import math

from sklearn.datasets import load_wine
from sklearn.model_selection import KFold
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve

%matplotlib inline
```

In [2]:
```python
features, labels = load_wine(return_X_y = True)
```

In [3]:
```python
features.shape
```

Out[3]: (178, 13)

In [4]:
```python
model_selection = KFold(n_splits = 3, random_state = 45, shuffle = True)
```

In [5]:
```python
model_selection.get_n_splits(features)
```

Out[5]: 3

k-Fold Cross-Validation

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. K refers to the number of groups that our data will be splited into.

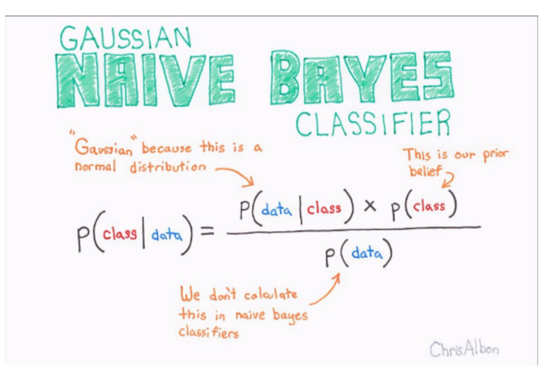
The process of k-fold is as as follows:

- Shuffle the dataset randomly.
- Split the data set into k parts.
- For each unique groups:
  - split the data to training and test set.
  - fir a model on the training set and evaluate it on the test set.
  - retain the evaluation result and remove the model.
- Estimate the performance of the model using the validation results.

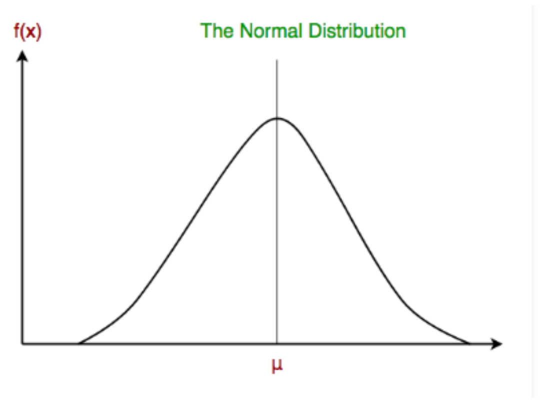
In [6]:
```python
for train_indx, test_indx in model_selection.split(features):
    x_train, x_test = features[train_indx], features[test_indx]
    y_train, y_test = labels[train_indx], labels[test_indx]
```

Guassian Naive Bayse

In Gaussian Naive Bayes, continuous values associated with each feature are assumed to be distributed according to a Gaussian distribution.



We can also call it normal disterbuation.



And the final equation which will be implemented in code is as follow:

$$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

In [7]:
```python
class GussianNB:
    def __init__(self, features, labels, x_test):
        self.features = features
        self.labels = labels
        self.num_class = len(np.unique(labels))
        self.classes = np.unique(labels)
        self.x_test = x_test
        self. prior = 1 / self.num_class

    def fit(self):
        features = pd.DataFrame(self.features)
        labels = pd.Series(self.labels)
        mean = features.groupby(by = labels).mean()
        variance = features.groupby(by = labels).var()

        return (mean.values, variance.values)

    def predict(self, mean, variance):
        mean_var = list()
        prediction_list = list()
        final_probs_list = list()

        for i in range(len(mean)):
            m_row = mean[i]
            v_row = variance[i]

            for index, value in enumerate(m_row):
                mean_val = value
                var_val = v_row[index]
                mean_var.append([mean_val, var_val])

        mean_var_arr = np.array(mean_var)
        separated_mean_var = np.vsplit(mean_var_arr, self.num_class)

        for k in range(len(self.x_test)):
            prob_list = list()
            final_prob = list()

            for i in range(self.num_class):
                array_class = separated_mean_var[i]

                for j in range(len(array_class)):
                    class_mean = array_class[j][0]
                    class_var = array_class[j][1]
                    x_values = self.x_test[k][j]

                    prob_list.append([self.gnb_equation(x_values, class_mean, class_var)])

            prob_array = np.array(prob_list)
            separated_prob = np.vsplit(prob_array, self.num_class)

            for i in separated_prob:
                class_prop = np.prod(i) * self.prior
                final_prob.append(class_prop)

            maximum_prob = max(final_prob)
            final_probs_list.append(maximum_prob)
            prop_max_index = final_prob.index(maximum_prob)
            prediction = self.classes[prop_max_index]
            prediction_list.append(prediction)

        return (prediction_list, final_probs_list)

    def gnb_equation(self, sample, mean, variance):
        e = np.e
        pi = np.pi

        equation_part_1 = 1 / np.sqrt(2 * pi * variance)
        equation_part_2 = np.exp(-((sample - mean))**2 / (2 * variance))
        final_equation = equation_part_1 * equation_part_2

        return final_equation
```

In [8]:
```python
model = GussianNB(x_train, y_train, x_test)
```

In [9]:
```python
mean, var = model.fit()
```

In [10]:
```python
predictions, probs_list = model.predict(mean, var)
```

In [11]:
```python
predictions_array = np.array(predictions)
probs_array = np.array(probs_list)
```

In [12]:
```python
accuracy_score(y_true = y_test, y_pred = predictions_array)
```

Out[12]: 0.9661016949152542

In [13]:
```python
cm = confusion_matrix(y_test, predictions_array)
```

In [14]:
```python
cm
```

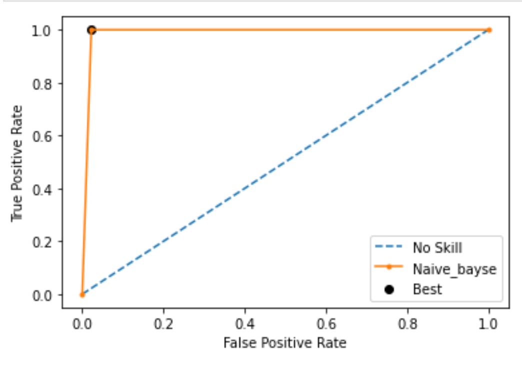
Out[14]:
```
array([[21,  0,  0],
       [ 1, 22,  1],
       [ 0,  0, 14]], dtype=int64)
```

In [25]:
```python
def self_calculated_metrics(cnf_matrix):
    FP = cnf_matrix.sum(axis=0) - np.diag(cnf_matrix)
    FN = cnf_matrix.sum(axis=1) - np.diag(cnf_matrix)
    TP = np.diag(cnf_matrix)
    TN = cnf_matrix.sum() - (FP + FN + TP)

    FP = FP.astype(float)
    FN = FN.astype(float)
    TP = TP.astype(float)
    TN = TN.astype(float)

    TPR = TP/(TP+FN)
    FPR = FP/(FP+TN)

    return (TPR, FPR)
```

In [26]:
```python
TPR, FPR = self_calculated_metrics(cm)
```

In [ ]:
```python
fpr, tpr, thresh = roc_curve(y_test, predictions_array, pos_label = 2)
```

In [48]:
```python
gmeans = np.sqrt(tpr * (1-fpr))
ix = np.argmax(gmeans)
plt.plot([0,1], [0,1], linestyle='--', label='No Skill')
plt.plot(fpr, tpr, marker='.', label='Naive_bayse')
plt.scatter(fpr[ix], tpr[ix], marker='o', color='black', label='Best')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```



In [28]:
```python
TPR
```

Out[28]: `array([1.        , 0.91666667, 1.        ])`

In [29]:
```python
FPR
```

Out[29]: `array([0.02631579, 0.        , 0.02222222])`

In [49]:
```python
fpr
```

Out[49]: `array([0.        , 0.02222222, 1.        ])`

In [50]:
```python
tpr
```

Out[50]: `array([0., 1., 1.])`

In [51]:
```python
thresh
```

Out[51]: `array([3, 2, 0])`

In [ ]: