Introduction to k-Nearest Neighbors (kNN) Algorithm

- The k-Nearest Neighbors (kNN) algorithm is a non-parametric, instance-based learning method used for classification and regression. It works by finding the k-nearest data points (neighbors) to a query point and making predictions based on the majority label (for classification) or the average value (for regression) of these neighbors. Due to its simplicity and effectiveness in many scenarios, kNN is one of the most widely used algorithms in machine learning.

kNN Intuition

- The core idea behind kNN is to make predictions based on the similarity of data points. This similarity is typically measured using distance metrics such as Euclidean distance. For a given query point, the algorithm:

1. Calculates the distance between the query point and all points in the training data.
2. Identifies the k-nearest neighbors based on these distances.
3. Predicts the output based on the majority class (in classification) or the average of the neighbors' values (in regression).

kNN on Breast Cancer Dataset with Code Example

How to Select k?

Choosing the optimal value of k is crucial for the performance of the kNN algorithm. A small value of k can lead to a noisy decision boundary (overfitting), while a large value of k can smooth out the decision boundary too much (underfitting). Common strategies to select k include:

- **Cross-Validation**: Using techniques like k-fold cross-validation to evaluate different values of k and select the one with the best performance.
- **Heuristics**: Often, (\sqrt{n}) (where (n) is the number of samples) is used as a heuristic for choosing k.
- **Empirical Testing**: Experimenting with a range of k values and observing the results on validation data.

In [1]:
```
import numpy as np
import pandas as pd
```

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

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430

5 rows × 33 columns

In [3]:
```
df.drop(columns=['id','Unnamed: 32'],inplace=True)
df.head()
```

Out[3]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_
0	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0
1	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0
2	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0
3	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0
4	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0

5 rows × 31 columns

In [4]:
```
df.shape
```

Out[4]: (569, 31)

In [5]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df.iloc[:,1:], df.iloc[:,0],test_size=0.2, random_state=2)
```

In [6]:
```
X_train.head()
```

Out[6]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fr
560	14.05	27.15	91.38	600.4	0.09929	0.11260	0.04462	0.04304	0.1537	
428	11.13	16.62	70.47	381.1	0.08151	0.03834	0.01369	0.01370	0.1511	
198	19.18	22.49	127.50	1148.0	0.08523	0.14280	0.11140	0.06772	0.1767	
203	13.81	23.75	91.56	597.8	0.13230	0.17680	0.15580	0.09176	0.2251	
41	10.95	21.35	71.90	371.1	0.12270	0.12180	0.10440	0.05669	0.1895	

5 rows × 30 columns

In [7]:
```
X_train.shape
```

Out[7]: (455, 30)

In [8]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

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

In [9]:
```
1 X_train
```

Out[9]:
```
array([[-0.01330339,  1.7757658 , -0.01491962, ..., -0.13236958,
        -1.08014517, -0.03527943],
       [-0.8448276 , -0.6284278 , -0.87702746, ..., -1.11552632,
        -0.85773964, -0.72098905],
       [ 1.44755936,  0.71180168,  1.47428816, ...,  0.87583964,
         0.4967602 ,  0.46321706],
       ...,
       [-0.46608541, -1.49375484, -0.53234924, ..., -1.32388956,
        -1.02997851, -0.75145272],
       [-0.50025764, -1.62161319, -0.527814  , ..., -0.0987626 ,
         0.35796577, -0.43906159],
       [ 0.96060511,  1.21181916,  1.00427242, ...,  0.8956983 ,
        -1.23064515,  0.50697397]])
```

In [10]:
```
1 X_train.shape
```

Out[10]:
```
(455, 30)
```

In [11]:
```
1 from sklearn.neighbors import KNeighborsClassifier
2
3 knn = KNeighborsClassifier(n_neighbors=3)
```

In [12]:
```
1 knn.fit(X_train,y_train)
```

Out[12]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

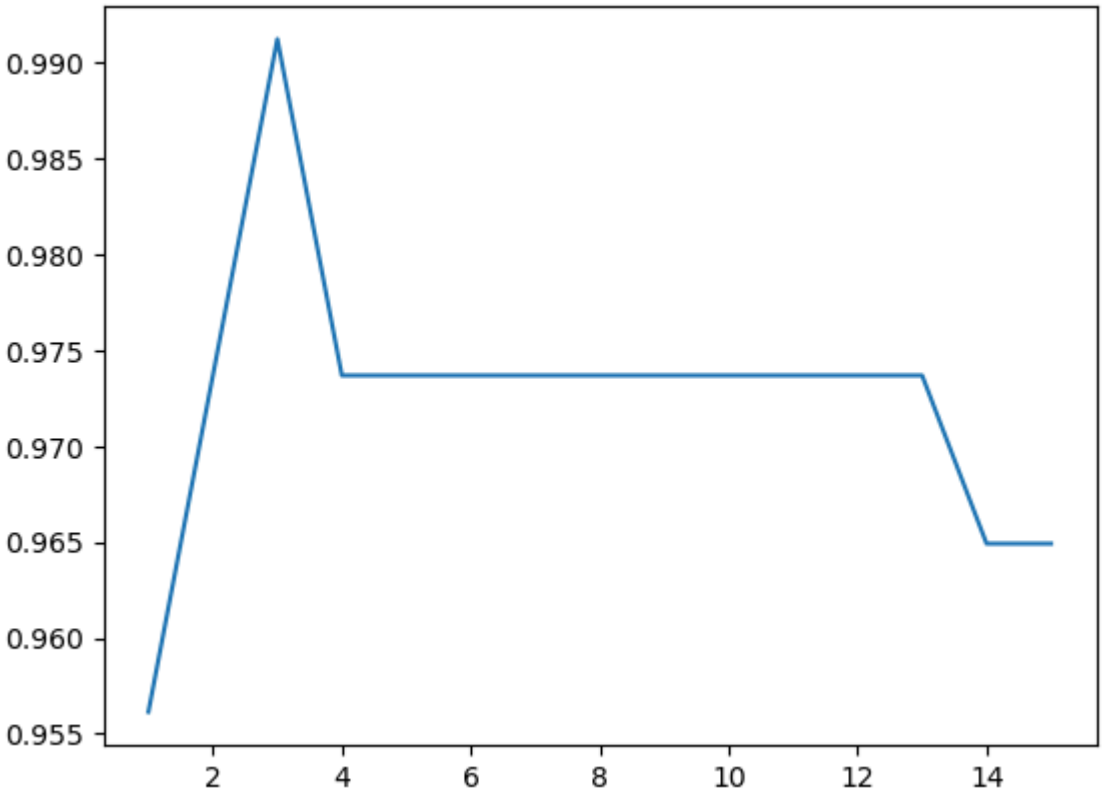
In [13]:
```
from sklearn.metrics import accuracy_score

y_pred = knn.predict(X_test)

accuracy_score(y_test, y_pred)
```

Out[13]: 0.9912280701754386

In [14]:
```
scores = []

for i in range(1,16):

    knn = KNeighborsClassifier(n_neighbors=i)

    knn.fit(X_train,y_train)

    y_pred = knn.predict(X_test)

    scores.append(accuracy_score(y_test, y_pred))

```

In [15]:
```
import matplotlib.pyplot as plt

plt.plot(range(1,16),scores)
```

Out[15]:
```
[<matplotlib.lines.Line2D at 0x26ec540ad10>]
```



Decision Surface

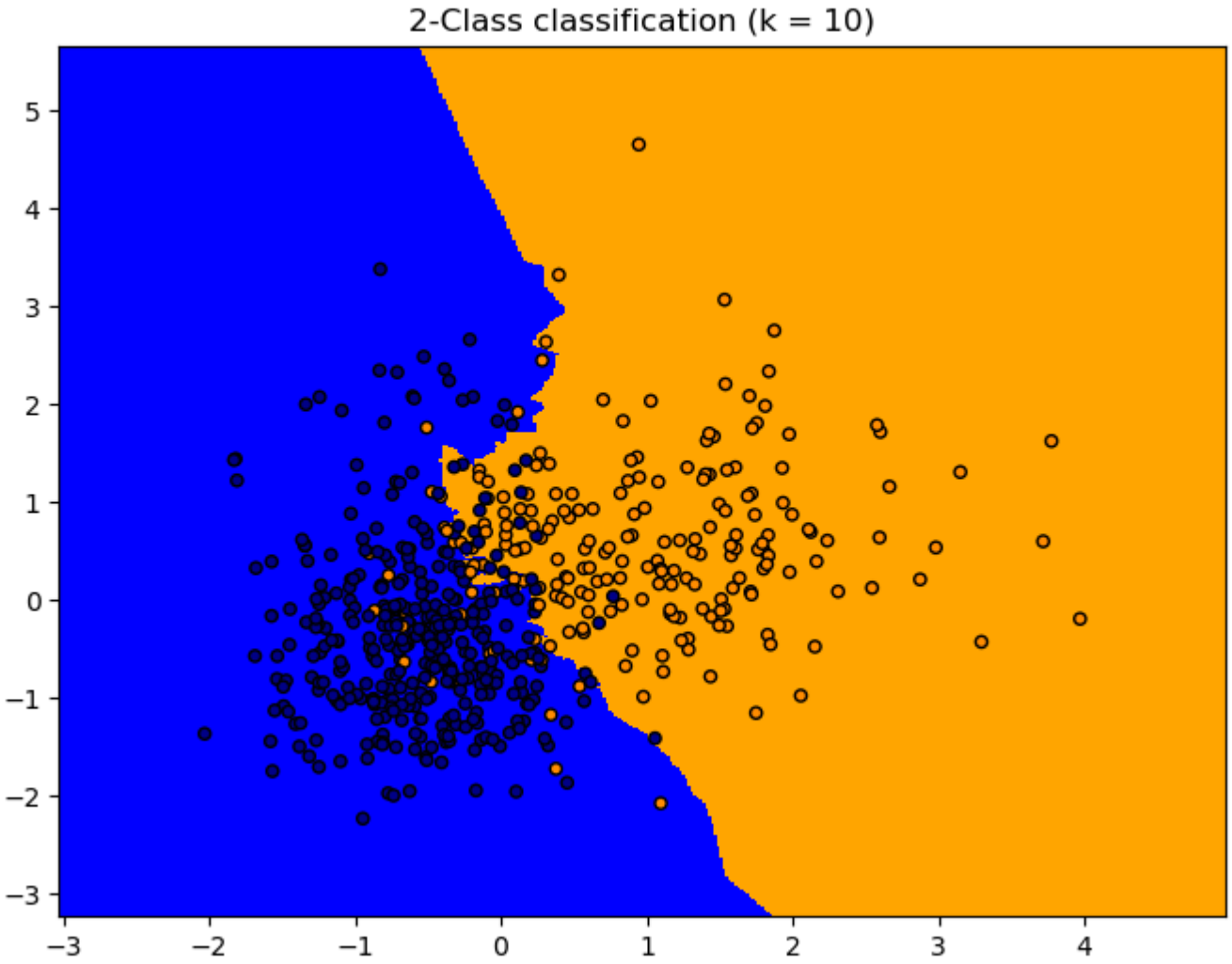
The decision surface of a kNN classifier represents how the algorithm divides the feature space into regions associated with different class labels. For low-dimensional data (1D or 2D), it can be visualized directly. For higher dimensions, techniques like PCA can reduce the dimensions for visualization purposes.

In [26]:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import neighbors, datasets
from sklearn.preprocessing import StandardScaler
from ipywidgets import interact, fixed

def load_data():
    cancer = datasets.load_breast_cancer()
    return cancer

def plot_decision_boundaries(n_neighbors, data, labels):
    h = .02
    cmap_light = ListedColormap(['orange', 'blue'])
    cmap_bold = ListedColormap(['darkorange', 'darkblue'])

    clf = neighbors.KNeighborsClassifier(n_neighbors)
    clf.fit(data, labels)

    x_min, x_max = data[:, 0].min() - 1, data[:, 0].max() + 1
    y_min, y_max = data[:, 1].min() - 1, data[:, 1].max() + 1

    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

    Z = Z.reshape(xx.shape)
    plt.figure(figsize=(8, 6))
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

    plt.scatter(data[:, 0], data[:, 1], c=labels, cmap=cmap_bold, edgecolor='k', s=20)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.title(f'2-Class classification (k = {n_neighbors})')
    plt.show()

cancer = load_data()

# Use only the first two features and standardize them.
X = StandardScaler().fit_transform(cancer.data[:, :2])
y = cancer.target

```

```
# Interactive widget
interact(plot_decision_boundaries, n_neighbors=(1, 20), data=fixed(X), labels=fixed(y));

```

n_neighbors 10



Overfitting and Underfitting in kNN

- **Overfitting**: When (k) is too small, the model may capture noise in the training data, leading to high variance and poor generalization to new data.
- **Underfitting**: When (k) is too large, the model becomes too generalized, failing to capture the underlying patterns in the data, leading to high bias and poor accuracy.

Limitations of kNN

1. **Computational Complexity**: kNN can be slow, especially for large datasets, because it requires distance calculations for all points during prediction.
2. **Curse of Dimensionality**: Performance degrades with high-dimensional data.
3. **Storage Requirements**: Requires storing the entire training dataset.
4. **Sensitivity to Irrelevant Features**: Irrelevant features can distort distance calculations.
5. **Imbalanced Data**: Struggles with imbalanced datasets.
6. **Choice of k and Distance Metric**: Performance is highly dependent on these choices.

Outro

The k-Nearest Neighbors algorithm is a powerful yet simple method for both classification and regression tasks. Its intuitive approach makes it easy to understand and implement, but it also comes with several limitations, particularly in terms of computational efficiency and sensitivity to data characteristics. By understanding these limitations and applying strategies such as feature selection, scaling, and cross-validation, one can effectively leverage kNN for a wide range of machine learning problems.

example 2

In [18]:

```
# Importing libraries and functions

import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import MinMaxScaler


# Load the dataset
data = pd.read_csv('glass.csv')
data.drop_duplicates(inplace=True)
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 213 entries, 0 to 213
Data columns (total 10 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   RI      213 non-null    float64
 1   Na      213 non-null    float64
 2   Mg      213 non-null    float64
 3   Al      213 non-null    float64
 4   Si      213 non-null    float64
 5   K       213 non-null    float64
 6   Ca      213 non-null    float64
 7   Ba      213 non-null    float64
 8   Fe      213 non-null    float64
 9   Type    213 non-null    int64  
dtypes: float64(9), int64(1)
memory usage: 18.3 KB
```

In [19]:
```
# Separate the features and target variable
X = data.drop('Type', axis=1)
y = data['Type']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [20]:
```
# Scaling Data

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

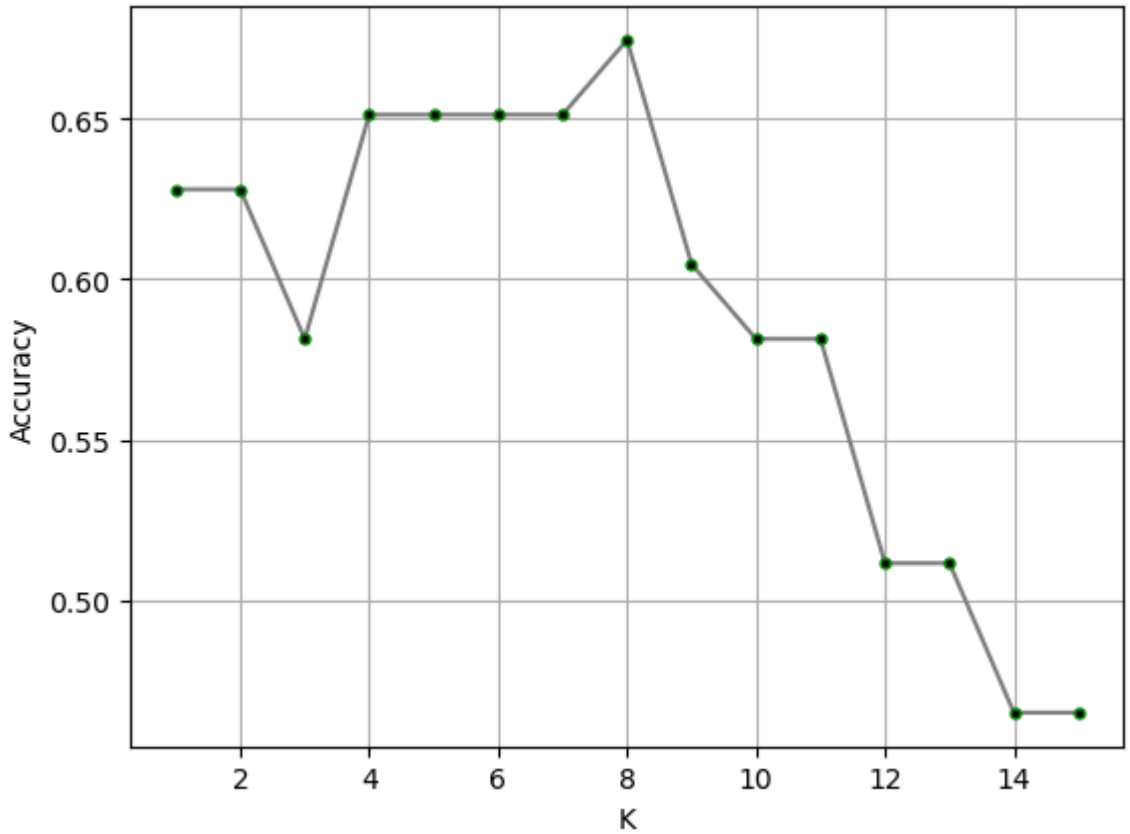
In [21]:
```
# Creating a model function

def knn_func(train_x, train_label, test_x, k):
    """
    train_x - train features
    train_label - train targets
    test_x - validation data(features)
    k - nearest neighbours <int>
    """
    knn = KNeighborsClassifier(n_neighbors = k)
    knn.fit(train_x, train_label)
    prediction = knn.predict(test_x)
    return prediction
```

In [22]:
```
# For best n_neighbours
import math
n = data.shape[0]
k_max = math.sqrt(n)
k_max
```

Out[22]: 14.594519519326424

In [23]:

```
normal_accuracy = []
k_values = range(1,16)

for k in k_values :
    y_pred = knn_func(X_train,y_train,X_test,k)
    accur = accuracy_score(y_test,y_pred)
    normal_accuracy.append(accur)

plt.plot(k_values,normal_accuracy,c="grey",marker=".",ms=7,mfc="black",mec="green")
plt.xlabel("K")
plt.ylabel("Accuracy")
plt.grid(True)
plt.show()
```



From above graph of Accuracy vs K, best value for n_beighbours is 8.

Let's go through the code step-by-step, explaining each part in detail with comments.

Import Necessary Libraries

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import neighbors, datasets
from sklearn.preprocessing import StandardScaler
from ipywidgets import interact, fixed
```

- **`numpy`** : A library for numerical operations on large, multi-dimensional arrays and matrices.
- **`matplotlib.pyplot`** : A plotting library for creating static, animated, and interactive visualizations in Python.
- **`ListedColormap` from `matplotlib.colors`** : Used to create custom color maps.
- **`sklearn`** : The `neighbors` module for kNN classifier and `datasets` for loading datasets.
- **`StandardScaler` from `sklearn.preprocessing`** : Standardizes features by removing the mean and scaling to unit variance.
- **`ipywidgets.interact` and `fixed`** : Used for creating interactive widgets in Jupyter notebooks.

Define Functions

Load Data Function

```
def load_data():
    cancer = datasets.load_breast_cancer()
    return cancer
```

- **`load_data`** : Loads the Breast Cancer Wisconsin dataset from `sklearn.datasets` .

Plot Decision Boundaries Function

```
def plot_decision_boundaries(n_neighbors, data, labels):
    h = .02  # Step size in the mesh
    cmap_light = ListedColormap(['orange', 'blue'])
    cmap_bold = ListedColormap(['darkorange', 'darkblue'])

    # Create an instance of Neighbors Classifier and fit the data.
    clf = neighbors.KNeighborsClassifier(n_neighbors)
    clf.fit(data, labels)

    # Determine the min and max values for the plot
    x_min, x_max = data[:, 0].min() - 1, data[:, 0].max() + 1
    y_min, y_max = data[:, 1].min() - 1, data[:, 1].max() + 1

    # Create a mesh grid
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

    # Predict the function value for the whole grid
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # Plot the contour and training points
    plt.figure(figsize=(8, 6))
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

    # Plot the training points
    plt.scatter(data[:, 0], data[:, 1], c=labels, cmap=cmap_bold, edgecolor='k', s=20)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.title(f'2-Class classification (k = {n_neighbors})')
    plt.show()
```

- **`plot_decision_boundaries`** :
  - **Parameters**:
    - `n_neighbors` : Number of neighbors to use in kNN.
    - `data` : Feature data (only two features are used).

    - `labels` : Target labels.
- **Variables**:
    - `h` : Step size in the mesh grid for plotting decision boundaries.
    - `cmap_light` and `cmap_bold` : Color maps for the plot.
- **Process**:
    - Create and fit a kNN classifier.
    - Determine the range of the plot.
    - Create a mesh grid over the feature space.
    - Predict the class for each point in the grid.
    - Plot the decision boundary and the training points.

Load the Dataset

```
cancer = load_data()
```

- **`cancer`** : Load the Breast Cancer Wisconsin dataset.

Standardize Features and Prepare Data

```
X = StandardScaler().fit_transform(cancer.data[:, :2])
y = cancer.target
```

- **`X`** : Standardize the first two features of the dataset. This ensures that each feature has a mean of 0 and a standard deviation of 1.
- **`y`** : Target labels.

Interactive Widget

```
interact(plot_decision_boundaries, n_neighbors=(1, 20), data=fixed(X), labels=fixed(y));
```

- **`interact`** : Creates an interactive widget that allows the user to change the number of neighbors (`n_neighbors`) dynamically.
- **Parameters**:
    - `plot_decision_boundaries` : Function to call when the widget changes.
    - `n_neighbors=(1, 20)` : Slider to choose a value for `n_neighbors` between 1 and 20.
    - `data=fixed(X)` : Fix the data argument to the standardized feature set.
    - `labels=fixed(y)` : Fix the labels argument to the target labels.

Full Explanation

This code creates an interactive visualization of the k-Nearest Neighbors classifier's decision boundary on a standardized version of the Breast Cancer Wisconsin dataset (using only the first two features). The `plot_decision_boundaries` function is called whenever the user changes the number of neighbors (`n_neighbors`) using the slider widget. The visualization helps to understand how the decision boundary changes with different values of `k` , highlighting areas where the classifier predicts different classes.