

## ✓ 1. Load Iris Plants Dataset

Reference: <https://www.kaggle.com/code>

```
!pip install --upgrade scikit-learn==1.0.2
!pip install --upgrade numpy==1.21.5
```

```
import time
import datetime
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('seaborn')

# Load dataset
from sklearn.datasets import load_iris
iris = load_iris()
iris_df = pd.DataFrame(iris.data, columns=iris.feature_names)
iris_df['target'] = iris.target
iris_df['target_names'] = [iris.target_names[t] for t in iris.target]
target_col = 'target'
feature_col = iris.feature_names
```

```
iris_df.head()
```

### Data Set Characteristics

**Number of Instances:** 150 (50 in each of three classes)

**Number of Attributes:** 4 numeric, predictive attributes and the class

### Attribute Information

- sepal length in cm
- sepal width in cm
- petal length in cm
- petal width in cm
- class:
  - Iris-Setosa
  - Iris-Versicolour
  - Iris-Virginica

## Summary Statistics

Features	Min	Max	Mean	SD	Class Correlation
sepal length	4.3	7.9	5.84	0.83	0.7826
sepal width	2.0	4.4	3.05	0.43	-0.4194
petal length	1.0	6.9	3.76	1.76	0.9490 (high!)
petal width	0.1	2.5	1.20	0.76	0.9565 (high!)

**Missing Attribute Values:** None

**Class Distribution:** 33.3% for each of 3 classes.

**Creator:** R.A. Fisher

**Donor:** Michael Marshall (MARSHALL%[PLU@io.arc.nasa.gov](mailto:PLU@io.arc.nasa.gov))

**Date:** July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken from Fisher's paper. Note that it's the same as in R, but not as in the UCI Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the pattern recognition literature. Fisher's paper is a classic in the field and is referenced frequently to this day. (See Duda & Hart, for example.) The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.

## References

- Fisher, R.A. "The use of multiple measurements in taxonomic problems" Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to Mathematical Statistics" (John Wiley, NY, 1950).
- Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis. (Q327.D83) John Wiley & Sons. ISBN 0-471-22361-1. See page 218.
- Dasarthy, B.V. (1980) "Nosing Around the Neighborhood: A New System Structure and Classification Rule for Recognition in Partially Exposed Environments". IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-2, No. 1, 67-71.
- Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule". IEEE Transactions on Information Theory, May 1972, 431-433.
- See also: 1988 MLC Proceedings, 54-64. Cheeseman et al's AUTOCLASS II conceptual clustering system finds 3 classes in the data.
- Many, many more ...

```
from sklearn.model_selection import train_test_split

X = iris_df[feature_col]
y = iris_df[target_col]

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=
```

## ✓ 2. K-Nearest Neighbors

For Classification: <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>

For Regression: <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html>

```
from sklearn.neighbors import KNeighborsClassifier
```



### Suggest Parameters



- **n\_neighbors:** 4
- **weights:** distance

```
n_neighbors = 5 #@param {type:"slider", min:1, max:15}
weights = 'distance' #@param ['uniform', 'distance']

knn = KNeighborsClassifier(
    n_neighbors=n_neighbors,
    weights=weights,
)

knn.fit(X_train, y_train)
```

**n\_neighbors:**  5 

**weights:** distance  

```
# Predict as Class
train_predict = knn.predict(X_train)
test_predict = knn.predict(X_test)

# Predict as Probability
train_predict_proba = knn.predict_proba(X_train)
```

```
# Show first 10 prediction
print(train_predict[:10])

# Probability of 0/1
print(train_predict_prob[:10])
```

```
# Actual Target
y_train[:10].tolist()
```

```
from sklearn.metrics import confusion_matrix

confusion_matrix(y_test, test_predict, labels=[0, 1])
```

```
from sklearn.metrics import classification_report

print(classification_report(y_test, test_predict , digits=4))
```

### ✓ 3. Visualization

Adapt from: [https://scikit-learn.org/stable/auto\\_examples/neighbors/plot\\_classification.html](https://scikit-learn.org/stable/auto_examples/neighbors/plot_classification.html)

```
#@markdown # **Select parameters to plot**
#@markdown ***(On simplified model for visualization - 2 variables)**
n_neighbors = 7 #@param {type:"slider", min:1, max:10, step:1}
weights = 'distance' #@param ['uniform', 'distance']
x_label = 'sepal length (cm)' #@param ['sepal length (cm)', 'sepal width (cm)',
y_label = 'sepal width (cm)' #@param ['sepal length (cm)', 'sepal width (cm)',
# - 2 variables)

x_idx = iris_df.columns.to_list().index(x_label)
y_idx = iris_df.columns.to_list().index(y_label)

from matplotlib import cm
from matplotlib.colors import ListedColormap
cmap_val = np.linspace(0.0, 1.0, 20)
cmap_light = ListedColormap(cm.get_cmap(plt.get_cmap('tab20'))(cmap_val)[:6:2])
cmap_bold = ListedColormap(cm.get_cmap(plt.get_cmap('tab20'))(cmap_val)[6:2])
h = .02 # step size in the mesh

model = KNeighborsClassifier(n_neighbors=n_neighbors, weights=weights)
model.fit(iris_df.iloc[:, [x_idx, y_idx]], iris_df[target_col])
x_min, x_max = iris_df.iloc[:, x_idx].min() - 1, iris_df.iloc[:, x_idx].max() +
v min. v max = iris df.iloc[:, v idx].min() - 1. iris df.iloc[:, v idx].max() +
```

**Select parameters to plot**  
**(On simplified model for visualization - 2 variables)**

**n\_neighbors:**  7

**weights:**

**x\_label:**

**y\_label:**

```

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

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(figsize=(8, 8), dpi=80)
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

# Plot also the training points
plt.scatter(
    iris_df.iloc[:, x_idx],
    iris_df.iloc[:, y_idx],
    c=iris_df[target_col],
    cmap=cmap_bold,
    edgecolor='k',
    s=60
)
plt.xlabel(x_label)
plt.xlim(xx.min(), xx.max())
plt.ylabel(y_label)
plt.ylim(yy.min(), yy.max())
plt.title("3-Class classification (k = %i, weights = '%s')"
          % (n_neighbors, weights))
plt.show()

```

## ✓ 4. Find the best K value

### ✓ 4.1 Grid Search

[https://scikit-learn.org/stable/modules/generated/sklearn.model\\_selection.GridSearchCV.html](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html)

```
from sklearn.model_selection import GridSearchCV
```

```
grid_search = GridSearchCV(
    estimator=KNeighborsClassifier(),
    param_grid=dict(
        n_neighbors=[1,2,3,4,5,6,7,8,9,10],
        weights=['uniform', 'distance'],
    ),
    scoring='f1_weighted',
    cv=5,
    n_jobs=-1 # Parallel
)

grid_start_time = time.time()
grid_search.fit(X_train, y_train)
grid_end_time = time.time()
print(f"Searching Time: {datetime.timedelta(seconds=grid_end_time-grid_start_time)}
```

```
# Get Searching Result
grid_search_result = grid_search.cv_results_
pd.DataFrame.from_dict(grid_search_result)
```

```
# Best Trained Model
model = grid_search.best_estimator_
```

```
# Predict with the best model
y_pred = model.predict(X_test)
y_pred[:10]
```

## ✓ 4.2 Randomized Search

[https://scikit-learn.org/stable/modules/generated/sklearn.model\\_selection.RandomizedSearchCV.html](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html)

```
from sklearn.model_selection import RandomizedSearchCV
```

```
rand_search = RandomizedSearchCV(  
    estimator=KNeighborsClassifier(),  
    param_distributions=dict(  
        n_neighbors=[1,2,3,4,5,6,7,8,9,10],  
        weights=['uniform', 'distance'],  
    ),  
    scoring='f1_weighted',  
    cv=5,  
    n_jobs=-1  
)  
  
rand_start_time = time.time()  
rand_search.fit(X_train, y_train)  
rand_end_time = time.time()  
print(f"Searching Time: {datetime.timedelta(seconds=rand_end_time-rand_start_ti
```

```
# Get Searching Result  
rand_search_result = rand_search.cv_results_  
pd.DataFrame.from_dict(rand_search_result)
```

```
# Best Trained Model  
model = rand_search.best_estimator_
```

```
# Predict with the best model  
y_pred = model.predict(X_test)  
y_pred[:10]
```