HW2 : KNN Classification and Cross Validation-10 Points

Undo delete cell ⌘/Ctrl+M Z
Redo ⌘/Ctrl+Shift+Y
Select all cells ⌘/Ctrl+Shift+A
Cut cell or selection
Copy cell or selection
Paste
Delete selected cells ⌘/Ctrl+M D
Find and replace ⌘/Ctrl+H
Find next ⌘/Ctrl+G
Find previous ⌘/Ctrl+Shift+G
Notebook settings
Clear all outputs

You have t IW

1. Firstl otebook)
2. Firstl

Import

Add any l is HW in the cell below.

```
%matplotl
from skle
import pa
import ma
import nu
from skle
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.model_selection import ShuffleSplit
```

Load Dataset

```
iris_dataset = load_iris()
```

Inspect Data

```
# check dataset type
type(iris_dataset)
```

```
sklearn.utils._bunch.Bunch
```

Bunch objects are sometimes used as an output for functions and methods. They extend dictionaries by enabling values to be accessed by key, bunch["value_key"], or by an attribute, bunch.value_key.

```
# check various keys to see what attrinbutes the data have
iris_dataset.keys()
```

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])
```

```
iris_dataset.DESCR
```

```
'.. _iris_dataset:\n\nIris plants dataset\n--------------------\n\n**Data Set Characteristics:**\n\n    :Number of Instances:
classes)\n    :Number of Attributes: 4 numeric, predictive attributes and the class\n    :Attribute Information:\n        - s
- sepal width in cm\n        - petal length in cm\n        - petal width in cm\n        - class:\n                - Iris-Setc
s-Versicolour\n                - Iris-Virginica\n                \n    :Summary Statistics:\n\n    ============== ==== ==== =
========\n                    Min  Max   Mean    SD   Class Correlation\n    ============== ==== ==== ======= ===== ========
ngth:   4.3  7.9   5.84   0.83    0.7826\n    sepal width:    2.0  4.4   3.05   0.43   -0.4194\n    petal length:   1.0  6.9
(high!)\n    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)\n    ============== ==== ==...'
```

```
# get independent variable(feature) names
iris_dataset.feature_names
```

```
['sepal length (cm)',
 'sepal width (cm)',
```

```
 'petal length (cm)',
 'petal width (cm)']
```

```
# get dependent variable(target) names
iris_dataset.target_names
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

```
# check the first five rows of the data (inmdependent variables)
iris_dataset.data[0:5]
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2]])
```

```
# check values for dependent variables
iris_dataset.target
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

```
# check the shape of the dataset
iris_dataset.data.shape
```

```
(150, 4)
```

Slpit data

Task 1 Split the data into Train/Test splits

Create X_train, X_test, y_train, y_test by splitting the data.

```
X = iris_dataset.data
y = iris_dataset.target
X_train, X_test, y_train, y_test = train_test_split(X , y, random_state = 42)
```

```
print(f"X_train shape {X_train.shape}")
print(f"y_train shape: {y_train.shape}")
```

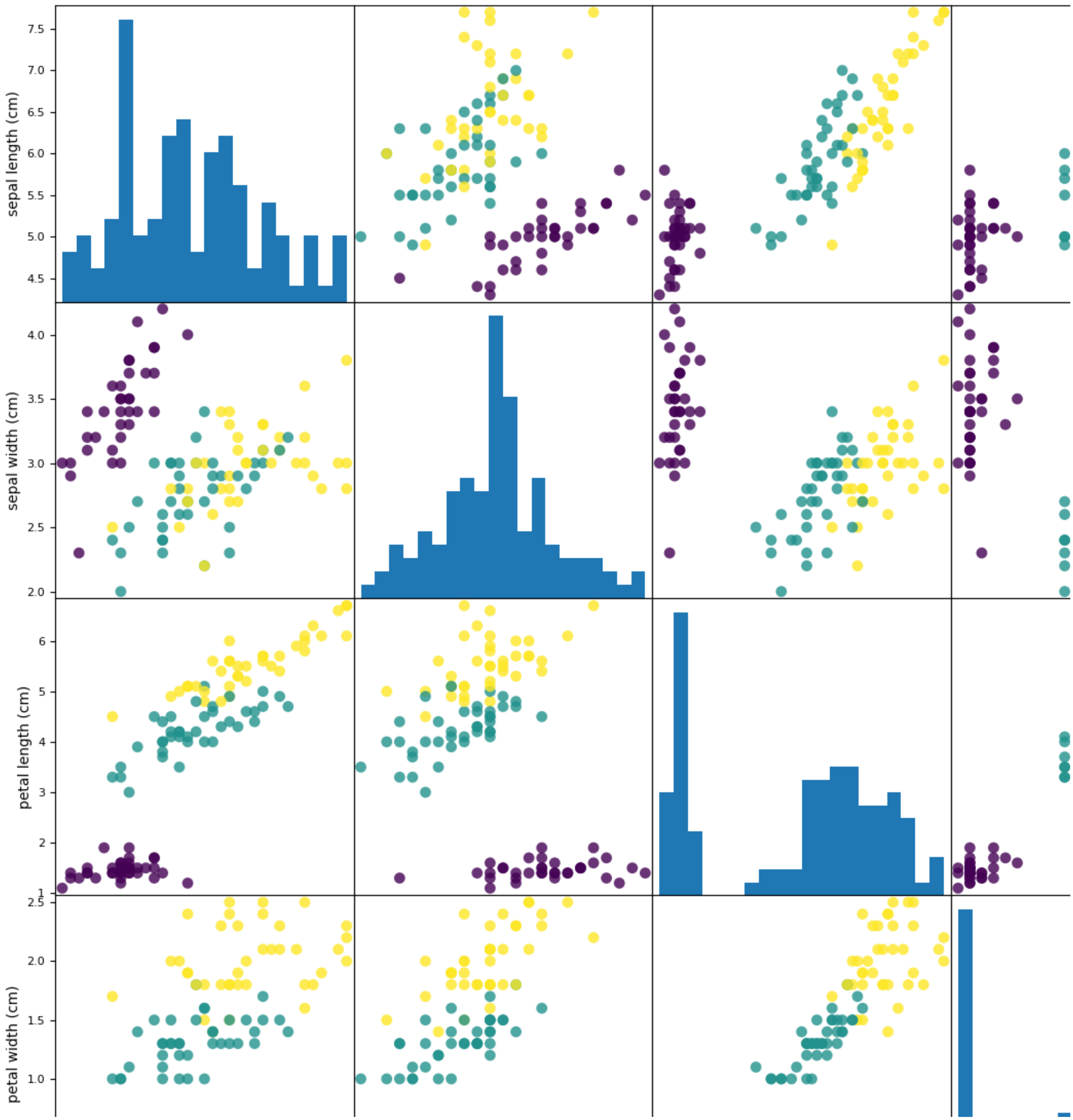
```
X_train shape (112, 4)
y_train shape: (112,)
```

Visualize data

```
# create dataframe from data in X_train
# label the columns using the strings in iris_dataset.feature_names
iris_dataframe = pd.DataFrame(X_train, columns=iris_dataset.feature_names)

# create a scatter matrix from the dataframe, color by y_train
pd.plotting.scatter_matrix(iris_dataframe, c=y_train, figsize=(15, 15),
                           marker='o', hist_kwds={'bins': 20}, s=60,
                           alpha=.8)
```

```
array([[<Axes: xlabel='sepal length (cm)', ylabel='sepal length (cm)'>,
        <Axes: xlabel='sepal width (cm)', ylabel='sepal length (cm)'>,
        <Axes: xlabel='petal length (cm)', ylabel='sepal length (cm)'>,
        <Axes: xlabel='petal width (cm)', ylabel='sepal length (cm)'>],
       [<Axes: xlabel='sepal length (cm)', ylabel='sepal width (cm)'>,
        <Axes: xlabel='sepal width (cm)', ylabel='sepal width (cm)'>,
        <Axes: xlabel='petal length (cm)', ylabel='sepal width (cm)'>,
        <Axes: xlabel='petal width (cm)', ylabel='sepal width (cm)'>],
       [<Axes: xlabel='sepal length (cm)', ylabel='petal length (cm)'>,
        <Axes: xlabel='sepal width (cm)', ylabel='petal length (cm)'>,
        <Axes: xlabel='petal length (cm)', ylabel='petal length (cm)'>,
        <Axes: xlabel='petal width (cm)', ylabel='petal length (cm)'>],
       [<Axes: xlabel='sepal length (cm)', ylabel='petal width (cm)'>,
        <Axes: xlabel='sepal width (cm)', ylabel='petal width (cm)'>,
        <Axes: xlabel='petal length (cm)', ylabel='petal width (cm)'>,
        <Axes: xlabel='petal width (cm)', ylabel='petal width (cm)'>]],
      dtype=object)
```




```
 # check the information for each column in the data
iris_dataframe.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 112 entries, 0 to 111
Data columns (total 4 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   sepal length (cm)  112 non-null    float64
 1   sepal width (cm)   112 non-null    float64
 2   petal length (cm)  112 non-null    float64
 3   petal width (cm)   112 non-null    float64
dtypes: float64(4)
memory usage: 3.6 KB
```

```
# check the summary statistics for the columns with numerical values
iris_dataframe.describe()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
count	112.000000	112.000000	112.000000	112.000000
mean	5.830357	3.040179	3.807143	1.214286
std	0.819123	0.437120	1.735310	0.747953
min	4.300000	2.000000	1.100000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.300000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.700000	4.200000	6.700000	2.500000

KNN Classification Model

Task 2- k-fold

- Fit the KNN Classifiation model on iris data.
- Finetune the hyperparameter -- n_neighbors using GridSearchCV.
- In GridSearchCV use **k-fold** (with 5 splits) for cross validation.
- Print the value of n_neighbors that gives the best fit.
- Print the best mean cross validation score.
- Use the trained model to get accuracy on train and test data sets.
- Print the accuracy for train and test data sets.

```
Knn_classification= KNeighborsClassifier()

param_grid_1 = {'n_neighbors':  np.arange(1, 16, 1)}


# specify the cross-validation startegy - use k-fold for this task

folds = KFold(n_splits=5, shuffle=True, random_state=0)

# Use GridSearchCV for kNN classification

grid_1 = GridSearchCV(Knn_classification, param_grid=param_grid_1, cv=folds,
                   return_train_score=True)

# Now fit the  GridSearchCV on the X_train, y_train by using fit() method

grid_1.fit(X_train, y_train)


print(f"best mean cross-validation score: {grid_1.best_score_}")
print(f"best parameters: {grid_1.best_params_}")

# We can check the accuracy score of training dataset and test dataset.
print(f"train-set score: {grid_1.score(X_train, y_train):.4f}")
print(f"test-set score: {grid_1.score(X_test, y_test):.4f}")
```

```
best mean cross-validation score: 0.9644268774703558
best parameters: {'n_neighbors': 8}
train-set score: 0.9643
test-set score: 1.0000
```

Task 3- Stratified k-fold

- Repeat Task 2 but use stratified k-fold for cross-validation.
- Stratified k-fold is the default strategy for GridSearchCV.
- We will again use 5 splits.

```
param_grid_2 = {'n_neighbors':  np.arange(1, 16, 1)}


# Use GridSearchCV for kNN classification
# We do not need to specify the croass validation startegy as stratified k-fold is the
# default strategy. Just use cv = # number of splits inside GridSearchCV

grid_2 = GridSearchCV(Knn_classification, param_grid = param_grid_2,cv=5,return_train_score=True)

# Now fit the  GridSearchCV on the X_train, y_train by using fit() method

grid_2.fit(X_train,y_train)

print(f"best mean cross-validation score: {grid_2.best_score_}")
print(f"best parameters: {grid_2.best_params_}")

# We can check the accuracy score of training dataset and test dataset.
print(f"train-set score: {grid_2.score(X_train,y_train):.4f}")
print(f"test-set score: {grid_2.score(X_test,y_test):.4f}")
```

```
best mean cross-validation score: 0.9644268774703558
best parameters: {'n_neighbors': 4}
train-set score: 0.9643
test-set score: 1.0000
```

Task 4- Stratified Shuffle Split

- Repeat Task 2 but use Stratified Shuffle Split in GridSearchCV.
- We will again use 5 splits.
- For each split we will use 40% of the data for train split and 40% of the data for validation/test split**

```
param_grid_3 = {'n_neighbors':  np.arange(1, 16, 1)}

# specify the cross-validation startegy - use Stratified Shuffle Split for this task
folds = ShuffleSplit(n_splits=5,test_size=0.4,random_state=0)

# Use GridSearchCV with kNN classification
grid_3 = GridSearchCV(Knn_classification,cv=folds,param_grid=param_grid_3,return_train_score=True)

# Now fit the  GridSearchCV on the X_train, y_train by using fit() method

grid_3.fit(X_train,y_train)

print(f"best mean cross-validation score: {grid_3.best_score_}")
print(f"best parameters: {grid_3.best_params_}")

# We can check the accuracy score of training dataset and test dataset.
print(f"train-set score: {(grid_3.score(X_train,y_train)):.4f}")
print(f"test-set score: {grid_3.score(X_test,y_test):.4f}")
```

```
best mean cross-validation score: 0.9555555555555555
best parameters: {'n_neighbors': 10}
train-set score: 0.9643
test-set score: 1.0000
```

Task 5- Predictions

- We will now make prediction on the new data.
- Use the model estimated in task3 to make prediction on the new dataset.

- Use model.predict(X_new) to make predictions on the new data.

```
X_new = np.array([[5, 2.9, 1, 0.2]])

prediction = grid_2.predict(X_new)
print(f"Prediction: {grid_2.score(X,y)}")
print(f"Predicted target name:{iris_dataset['target_names'][prediction]}")
```

```
Prediction: 0.9733333333333334
Predicted target name:['setosa']
```

Task 6

In the text cell below, briefly explain when you will use the following cross validation strategies

- k-fold
- stratified k-fold
- shuffle and split
- Startified shuffle and split
- leave one out

TYPE YOUR ANSWER HERE

```
ides entire data into no of splits you want(k=n, if k =5, conducts training & test 5 times with 80% in training data and 20% in te
k-fold: Same as K-fold but data is split in a way that each time tsting data has n% proportion of each category(i.e. the target va
plit: When dividing the test and train, the data shuffles everytime when selection of other 20%. Generally when we have a large ba
Shuffle & split: same as Shuffle & split but maintains the proportion w.r.t. classes distribution in the data. Used when large imb
ut : taking new single sample as a testing data in iteratio, untill all data has a chance to be testing data.Used when we have sma
```

Double-click (or enter) to edit

✓ 0s completed at 7:41 PM