k-Nearest Neighbors Classifier

In this notebook, you will implement your own k-nearest neighbors (k-NN) algorithm for the classification problem. You are supposed to learn:

- How to prepare the dataset for "training" and testing of the model.
- How to implement k-nearest neighbors classification algorithm.
- How to evaluate the performance of your classifier.

Instructions:

- Read carefuly through this notebook. Be sure you understand what is provided to you, and what is required from you.
- Place your code only in sections annotated with `### START CODE HERE ###` and `### END CODE HERE ###`.
- Use comments whenever the code is not self-explanatory.
- Submit an executable notebook (`*.ipynb`) with your solution to BlackBoard.

Enjoy :-)

Packages

Following packages is all you need. Do not import any additional packages!

- Pandas is a library providing easy-to-use data structures and data analysis tools.
- Numpy library provides support for large multi-dimensional arrays and matrices, along with functions to operate on these.

```
import pandas as pd
import numpy as np
```

Problem

You are given a dataset `mushrooms.csv` with characteristics/attributes of mushrooms, and your task is to implement and evaluate a k-nearest neighbors classifier able to say whether a mushroom is poisonous or edible based on its attributes.

Dataset

The dataset of mushroom characteristics is freely available at Kaggle Datasets where you can find further information about the dataset. It consists of 8124 mushrooms characterized by 23 attributes (including the class). Following is the overview of attributes and values:

- class: edible=e, poisonous=p
- cap-shape: bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s
- cap-surface: fibrous=f,grooves=g,scaly=y,smooth=s

- cap-color: brown=n,buff=b,cinnamon=c,gray=g,green=r,pink=p,purple=u,red=e,white=w,yellow=y
- bruises: bruises=t,no=f
- odor: almond=a,anise=l,creosote=c,fishy=y,foul=f,musty=m,none=n,pungent=p,spicy=s
- gill-attachment: attached=a,descending=d,free=f,notched=n
- gill-spacing: close=c,crowded=w,distant=d
- gill-size: broad=b,narrow=n
- gill-color: black=k,brown=n,buff=b,chocolate=h,gray=g, green=r,orange=o,pink=p,purple=u,red=e,white=w,yellow=y
- stalk-shape: enlarging=e,tapering=t
- stalk-root: bulbous=b,club=c,cup=u,equal=e,rhizomorphs=z,rooted=r,missing=?
- stalk-surface-above-ring: fibrous=f,scaly=y,silky=k,smooth=s
- stalk-surface-below-ring: fibrous=f,scaly=y,silky=k,smooth=s
- stalk-color-above-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y
- stalk-color-below-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y
- veil-type: partial=p,universal=u
- veil-color: brown=n,orange=o,white=w,yellow=y
- ring-number: none=n,one=o,two=t
- ring-type: cobwebby=c,evanescent=e,flaring=f,large=l,none=n,pendant=p,sheathing=s,zone=z
- spore-print-color: black=k,brown=n,buff=b,chocolate=h,green=r,orange=o,purple=u,white=w,yellow=y
- population: abundant=a,clustered=c,numerous=n,scattered=s,several=v,solitary=y
- habitat: grasses=g,leaves=l,meadows=m,paths=p,urban=u,waste=w,woods=d

Let's load the dataset into so called Pandas dataframe.

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
import os
os.chdir('./gdrive/MyDrive/Zou/KNN')
```

```
mushrooms_df = pd.read_csv('mushrooms.csv')
```

Now we can take a closer look at the data.

```
mushrooms_df
```

```
     class cap-shape cap-surface cap-color bruises odor gill-
attachment  \
0        p         x           s         n       t    p
f
1        e         x           s         y       t    a
f
2        e         b           s         w       t    l
f
3        p         x           y         w       t    p
f
4        e         x           s         g       f    n
f
...    ...       ...         ...       ...     ...  ...              ..
.
8119     e         k           s         n       f    n
a
8120     e         x           s         n       f    n
a
8121     e         f           s         n       f    n
a
8122     p         k           y         n       f    y
f
8123     e         x           s         n       f    n
a

     gill-spacing gill-size gill-color  ... stalk-surface-below-
ring  \
0               c         n          k  ...                        s

1               c         b          k  ...                        s

2               c         b          n  ...                        s

3               c         n          n  ...                        s

4               w         b          k  ...                        s

...           ...       ...        ...  ...                      ...

8119            c         b          y  ...                        s

8120            c         b          y  ...                        s

8121            c         b          n  ...                        s

8122            c         n          b  ...                        k

8123            c         b          y  ...                        s


     stalk-color-above-ring stalk-color-below-ring veil-type veil-
```

```
color  \
0                        w                      w         p
w
1                        w                      w         p
w
2                        w                      w         p
w
3                        w                      w         p
w
4                        w                      w         p
w
...                    ...                    ...       ...         ..
.
8119                     o                      o         p
o
8120                     o                      o         p
n
8121                     o                      o         p
o
8122                     w                      w         p
w
8123                     o                      o         p
o

     ring-number ring-type spore-print-color population habitat
0              o         p                 k          s       u
1              o         p                 n          n       g
2              o         p                 n          n       m
3              o         p                 k          s       u
4              o         e                 n          a       g
...          ...       ...               ...        ...     ...
8119           o         p                 b          c       l
8120           o         p                 b          v       l
8121           o         p                 b          c       l
8122           o         e                 w          v       l
8123           o         p                 o          c       l

[8124 rows x 23 columns]
```

You can also print an overview of all attributes with the counts of unique values.

```
mushrooms_df.describe().T
```

```
                         count unique top  freq
class                     8124      2   e  4208
cap-shape                 8124      6   x  3656
cap-surface               8124      4   y  3244
cap-color                 8124     10   n  2284
bruises                   8124      2   f  4748
odor                      8124      9   n  3528
gill-attachment           8124      2   f  7914
```

```
gill-spacing              8124      2    c   6812
gill-size                 8124      2    b   5612
gill-color                8124     12    b   1728
stalk-shape               8124      2    t   4608
stalk-root                8124      5    b   3776
stalk-surface-above-ring  8124      4    s   5176
stalk-surface-below-ring  8124      4    s   4936
stalk-color-above-ring    8124      9    w   4464
stalk-color-below-ring    8124      9    w   4384
veil-type                 8124      1    p   8124
veil-color                8124      4    w   7924
ring-number               8124      3    o   7488
ring-type                 8124      5    p   3968
spore-print-color         8124      9    w   2388
population                8124      6    v   4040
habitat                   8124      7    d   3148
```

The dataset is pretty much balanced. That's a good news for the evaluation.

Dataset Preprocessing

As our dataset consist of nominal/categorical values only, we will encode the strings into integers which will allow us to use similiraty measures such as Euclidean distance.

```python
def encode_labels(df):
    import sklearn.preprocessing
    encoder = {}
    for col in df.columns:
        le = sklearn.preprocessing.LabelEncoder()
        le.fit(df[col])
        df[col] = le.transform(df[col])
        encoder[col] = le
    return df, encoder

mushrooms_encoded_df, encoder = encode_labels(mushrooms_df)
```

```python
mushrooms_encoded_df
```

```
      class  cap-shape  cap-surface  cap-color  bruises  odor  \
0         1          5            2          4        1     6
1         0          5            2          9        1     0
2         0          0            2          8        1     3
3         1          5            3          8        1     6
4         0          5            2          3        0     5
...     ...        ...          ...        ...      ...   ...
8119      0          3            2          4        0     5
8120      0          5            2          4        0     5
8121      0          2            2          4        0     5
8122      1          3            3          4        0     8
8123      0          5            2          4        0     5
```

```
      gill-attachment  gill-spacing  gill-size  gill-color  ...  \
0                   1             0          1           4  ...   
1                   1             0          0           4  ...   
2                   1             0          0           5  ...   
3                   1             0          1           5  ...   
4                   1             1          0           4  ...   
...               ...           ...        ...         ...  ...   
8119                0             0          0          11  ...   
8120                0             0          0          11  ...   
8121                0             0          0           5  ...   
8122                1             0          1           0  ...   
8123                0             0          0          11  ...   

      stalk-surface-below-ring  stalk-color-above-ring  \
0                            2                       7   
1                            2                       7   
2                            2                       7   
3                            2                       7   
4                            2                       7   
...                        ...                     ...   
8119                         2                       5   
8120                         2                       5   
8121                         2                       5   
8122                         1                       7   
8123                         2                       5   

      stalk-color-below-ring  veil-type  veil-color  ring-number  
ring-type  \
0                          7          0           2            1  
4   
1                          7          0           2            1  
4   
2                          7          0           2            1  
4   
3                          7          0           2            1  
4   
4                          7          0           2            1  
0   
...                      ...        ...         ...          ...  
...   
8119                       5          0           1            1  
4   
8120                       5          0           0            1  
4   
8121                       5          0           1            1  
4   
8122                       7          0           2            1  
0   
8123                       5          0           1            1  
4   
```

```
      spore-print-color  population  habitat
0                     2           3        5
1                     3           2        1
2                     3           2        3
3                     2           3        5
4                     3           0        1
...                 ...         ...      ...
8119                  0           1        2
8120                  0           4        2
8121                  0           1        2
8122                  7           4        2
8123                  4           1        2

[8124 rows x 23 columns]
```

Dataset Splitting

Before we start with the implementation of our k-nearest neighbors algorithm we need to prepare our dataset for the "training" and testing.

First, we divide the dataset into attributes (often called features) and classes (often called targets). Keeping attributes and classes separately is a common practice in many implementations. This should simplify the implementation and make the code understandable.

```
X_df = mushrooms_encoded_df.drop('class', axis=1)  # attributes
y_df = mushrooms_encoded_df['class']  # classes
X_array = X_df.to_numpy()
y_array = y_df.to_numpy()
```

And this is how it looks like.

Next, we need to split the attributes and classes into training sets and test sets.

Exercise:

Implement the holdout splitting method with shuffling.

```
def train_test_split(X, y, test_size=0.2):
    import sys
    """
    Shuffles the dataset and splits it into training and test sets.

    :param X
        attributes
    :param y
        classes
    :param test_size
        float between 0.0 and 1.0 representing the proportion of the
dataset to include in the test split
```

```
    :return
        train-test splits (X-train, X-test, y-train, y-test)
    """
    ### START CODE HERE ###
    if test_size < 0 or test_size > 1:
      sys.exit("test_size should be between 0.0 and 1.0")

    X_copy = np.copy(X)
    y_copy = np.copy(y)

    np.random.shuffle(X_copy)
    np.random.shuffle(y_copy)

    nb_test_rows = round(X.shape[0] * test_size)

    X_test = X_copy[:nb_test_rows]
    X_train = X_copy[nb_test_rows:]

    y_test = y_copy[:nb_test_rows]
    y_train = y_copy[nb_test_rows:]
    ### END CODE HERE ###
    return X_train, X_test, y_train, y_test
```

Let's split the dataset into training and validation/test set with 67:33 split.

```
X_train, X_test, y_train, y_test = train_test_split(X_array, y_array,
0.33)
```

A quick sanity check...

```
assert len(X_train) == len(y_train)
assert len(y_train) == 5443
assert len(X_test) == len(y_test)
assert len(y_test) == 2681
```

Algorithm

The k-nearest neighbors algorithm doesn't require a training step. The class of an unseen sample is deduced by comparison with samples of known class.

Exercise:

Implement the k-nearest neighbors algorithm.

```
# Use this section to place any "helper" code for the `knn()`
function.

### START CODE HERE ###

def euclidian_distance(x, X):
  return np.sqrt(np.sum((x - X) ** 2, axis=1))
```

```
def most_common(x):
  # values : the unique values of x
  # counts : the number of occurrencies of each unique value
  values, counts = np.unique(x, return_counts=True)

  # np.argsort(counts)              : return the indexes that sort
counts
  # np.argsort(counts)[-1]          : to get the index of the most
frequent value
  # values[np.argsort(counts)[-1]]  : to get its value
  return values[np.argsort(counts)[-1]]

### END CODE HERE ###

def knn(X_true, y_true, X_pred, k=5):
    """
    k-nearest neighbors classifier.

    :param X_true
        attributes of the groung truth (training set)
    :param y_true
        classes of the groung truth (training set)
    :param X_pred
        attributes of samples to be classified
    :param k
        number of neighbors to use
    :return
        predicted classes
    """
    ### START CODE HERE ###

    y_pred = []
    for x in X_pred:
      distances = euclidian_distance(x, X_true)
      k_idx = np.argsort(distances)[:k]
      k_nearest_neighbors = y_true[k_idx]
      y_pred.append(most_common(k_nearest_neighbors))

    ### END CODE HERE ###
    return y_pred

y_hat = knn(X_train, y_train, X_test, k=5)
```

First ten predictions of the test set.

```
y_hat[:10]
```

```
[1, 0, 0, 0, 1, 0, 1, 0, 0, 1]
```

Evaluation

Now we would like to assess how well our classifier performs.

Exercise:

Implement a function for calculating the accuracy of your predictions given the ground truth and predictions.

```
def evaluate(y_true, y_pred):
    """
    Function calculating the accuracy of the model on the given data.

    :param y_true
        true classes
    :paaram y
        predicted classes
    :return
        accuracy
    """
    ### START CODE HERE ###

    accuracy = np.mean(y_true == y_pred)

    ### END CODE HERE ###
    return accuracy
```

```
accuracy = evaluate(y_test, y_hat)
print('accuracy =', accuracy)
```

```
accuracy = 0.4856396866840731
```

How many items where misclassified?

```
print('misclassified =', sum(abs(y_hat - y_test)))
```

```
misclassified = 1292
```

How balanced is our test set?

```
np.bincount(y_test)
```

```
array([1410, 1271])
```

If it's balanced, we don't have to be worried about objectivity of the accuracy metric.

Congratulations! At this point, hopefully, you have successufuly implemented a k-nearest neighbors algorithm able to classify unseen samples with high accuracy.

✌