Problem Set 1

Due: AUGUST 27, before class (before 4:00 PM).

How to submit

You need to send the `.ipynb` file with your answers plus an `.html` file, which will serve as a backup for us in case the `.ipynb` file cannot be opened on my or the TA's computer. In addition, you may also export the notebook as PDF and attach it to the email as well.

Please use the following subject header for sending in your homework, so that we can make sure that nothing gets lost:

Your id mentioned on offer letter: Your Name
.

Note No help will be provided bny instructor you have to do it on your own.

you will get certificates on basis its result

In [33]:
```
%load_ext watermark
%watermark  -d -u -a 'Khawar Amin, ID: GV00251' -v -p numpy,scipy,matplotlib,sklearn
```

```
The watermark extension is already loaded. To reload it, use:
  %reload_ext watermark
Author: Khawar Amin, ID: GV00251

Last updated: 2024-08-26

Python implementation: CPython
Python version       : 3.11.9
IPython version      : 8.20.0

numpy     : 1.26.4
scipy     : 1.12.0
matplotlib: 3.6.3
sklearn   : 1.4.2
```

The watermark package that is being used in the next code cell provides a helper function of the same name, `%watermark` for showing information about your computational environment. This is usefult to keep track of what software versions are/were being used. If you should encounter issues with the code, please make sure that your software package have the same version as the the ones shown in the pre-executed watermark cell.

Before you execute the watermark cell, you need to install watermark first. If you have not done this yet. To install the watermark package, simply run

```
!pip install watermark
```

or

```
!conda install watermark -c conda-forge
```

in the a new code cell. Alternatively, you can run either of the two commands (the latter only if you have installed Anaconda or Miniconda) in your command line terminal (e.g., a Linux shell, the Terminal app on macOS, or Cygwin, Putty, etc. on Windows).

For more information installing Python, please refer to the previous lectures and ask the TA for help.

E 1)

Pick 3 machine learning application examples from the first lecture (see section 1.2 in the lecture notes, shared in group and answer the following questions:

- What is the overall goal?
- How would an appropriate dataset look like?
- Which general machine learning category (supervised, unsupervised, reinforcement learning) does this problem fit in?
- How would you evaluate the performance of your model (in very general, non technical terms)

Example -- Email Spam classification:

- **Goal.** The goal is to classify incoming emails as either "spam" or "non-spam" (ham) to help users filter unwanted or potentially harmful messages from their inbox.

- **Dataset.** The dataset would consist of a large collection of emails with labels indicating whether each email is spam or not. The emails would contain various features such as the subject line, body text, sender address, attachments, and metadata like the time of receipt.

- **Category.** This problem falls under supervised learning because the model learns to classify emails based on labeled examples of spam and non-spam emails.

- **Measure Performance.** Performance can be evaluated by checking how accurately the model predicts whether new, unseen emails are spam or non-spam. Common metrics include accuracy, precision, recall, and F1-score. Specifically, the precision and recall of the spam class are important, as you want to minimize false positives (non-spam emails flagged as spam) and false negatives (spam emails that go undetected). **Example -- Drug Design:**

- **Goal.** The goal is to predict the effectiveness and potential side effects of new drug compounds before they are tested in clinical trials.
- **Dataset.** The dataset would consist of chemical structures of compounds, biological activity data, and associated outcomes (e.g., whether the compound is effective or has side effects).
- **Category.** This problem typically falls under supervised learning because it involves learning from labeled data (effective/ineffective, safe/unsafe).
- **Measure Performance.** The performance can be evaluated by checking how accurately the model predicts the effectiveness and safety of new compounds compared to known outcomes. This could be measured by accuracy, precision, recall, and possibly the area under the ROC curve. **Example -- Self Driving car:**
- **Goal.** The goal is to enable a car to navigate autonomously, recognizing and responding to road conditions, traffic signals, and obstacles.
- **Dataset.** The dataset would include images, videos, and sensor data (like LiDAR) from various driving environments, along with annotations like road boundaries, traffic signals, and labels for obstacles.
- **Category.** This problem fits into both supervised learning (for tasks like object detection) and reinforcement learning (for learning driving strategies and decision-making).
- **Measure Performance.** Performance can be evaluated based on the car's ability to safely and effectively navigate roads without human intervention, measured by metrics like collision rate, successful trip completion, and adherence to traffic laws.

< Double click on this cell to edit it and write your answers. Then press Shift+Enter to leave the editing mode. >

E 2)

If you think about the task of spam classification more thoroughly, do you think that the classification accuracy or misclassification error is a good error metric of how good an email classifier is? What are potential pitfalls? (Hint: think about false positives [non-spam email classified as spam] and false negatives [spam email classified as non-spam]).

No, classification accuracy alone isn't sufficient for evaluating a spam classifier. It can be misleading due to class imbalance. The key pitfalls include:False positive and false negative

E 3)

In the exercise example of E 1), email spam classification was listed as an example of a supervised machine learning problem. List 2 examples of unsupervised learning tasks that would fall into the category of clustering. In one or more sentences, explain why you would describe these examples as clustering tasks and not supervised learning tasks. Select examples that are not already that are in the "Lecture note list" from E 1).

Customer Segmentation:

Grouping customers based on purchasing behavior or demographics. This is a clustering task because the goal is to identify natural groupings in the data without predefined labels. Document Clustering:

Organizing a collection of documents into clusters based on content similarity. This is a clustering task because it involves finding structure in the data by grouping similar documents together, without using any labeled data.

E 4)

In the *k*-nearest neighbor (*k*-NN) algorithm, what computation happens at training and what computation happens at test time? Explain your answer in 1-2 sentences.

In the k-nearest neighbor (k-NN) algorithm, training involves storing the entire training dataset without any explicit model fitting. At test time, the algorithm computes the distance between the test instance and all training instances to identify the k-nearest neighbors and make predictions based on their labels.

E 5)

Does (*k*-NN) work better or worse if we add more information by adding more feature variables (assuming the number of training examples is fixed)? Explain your reasoning.

Adding more features can worsen k-NN performance due to the curse of dimensionality, making distances less meaningful.

E 6)

If your dataset contains several noisy examples (or outliers), is it better to increase or decrease *k*? Explain your reasoning.

Increasing k in the k-nearest neighbors algorithm helps to mitigate the influence of noisy examples and outliers by considering more neighbors when making a prediction. This averaging effect reduces the likelihood that a single noisy or outlier data point will skew the result. With a larger k, the decision is based on a broader range of neighbors, leading to more stable and robust predictions.< Double click on this cell to edit it and write your answers. Then press Shift+Enter to leave the editing mode. >

E 7)

Implement the Kronecker Delta function in Python,

$$\delta(i,j) = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j. \end{cases}$$

The `assert` statements are here to help you: They will raise an `AssertionError` if your function returns unexpected results based on the test cases.

In [3]:

```
# This is an example implementing a Dirac Delta Function

def dirac_delta(x):
    if x > 0.5:
        return 1
    else:
        return 0

assert dirac_delta(1) == 1
assert dirac_delta(2) == 1
assert dirac_delta(-1) == 0
assert dirac_delta(0.5) == 0
```

In [4]:

```
def kronecker_delta(i, j):

    return 1 if i == j else 0

# DO NOT EDIT THE LINES BELOW
assert kronecker_delta(1, 0) == 0
assert kronecker_delta(2, 2) == 1
assert kronecker_delta(-1, 1) == 0
assert kronecker_delta(0.5, 0.1) == 0
```

E 8)

Suppose `y_true` is a list that contains true class labels, and `y_pred` is an array with predicted class labels from some machine learning task. Calculate the prediction accuracy in percent (without using any external libraries).

In [5]:

```
y_true = [1, 2, 0, 1, 1, 2, 3, 1, 2, 1]
y_pred = [1, 2, 1, 1, 1, 0, 3, 1, 2, 1]

correct_predictions = sum(1 for true, pred in zip(y_true, y_pred) if true == pred)
accuracy = (correct_predictions / len(y_true)) * 100
```

```python
# Print the accuracy
print('Accuracy: %.2f%%' % accuracy)
```

```
Accuracy: 80.00%
```

E 9)

Import the NumPy library to create a 3x3 matrix with values ranging 0-8. The expected output should look as follows:

```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

In [6]:
```python
import numpy as np

matrix = np.arange(9).reshape(3, 3)


print(matrix)
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

E 10)

Use create a 2x2 NumPy array with random values drawn from a standard normal distribution using the random seed `123`:

If you are using the 123 random seed, the expected result should be:

```
array([[-1.0856306 ,  0.99734545],
       [ 0.2829785 , -1.50629471]])
```

In [7]:
```python
np.random.seed(123)

array = np.random.normal(loc=0, scale=1, size=(2, 2))


print(array)
```

```
[[-1.0856306   0.99734545]
 [ 0.2829785  -1.50629471]]
```

E 11)

Given an array `A`,

```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
```

```
        [ 9, 10, 11, 12],
        [13, 14, 15, 16]])
```

use the NumPy slicing syntax to only select the 2x2 center of this matrix, i.e., the subarray

```
array([[ 6,  7],
        [10, 11]]).
```

In [8]:

```
import numpy as np

A = np.array([
    [1, 2, 3, 4],
    [5, 6, 7, 8],
    [9, 10, 11, 12],
    [13, 14, 15, 16]
])

center_subarray = A[1:3, 1:3]
print(center_subarray)
```

```
[[ 6  7]
 [10 11]]
```

E 12)

Given the array `A` below, find the most frequent integer in that array:

In [11]:

```
import numpy as np

rng = np.random.RandomState(123)
A = rng.randint(0, 10, 200)
counts = np.bincount(A)
most_frequent = np.argmax(counts)

print("Most frequent integer:", most_frequent)
```

```
Most frequent integer: 3
```

E 13)

Complete the line of code below to read in the `'train_data.txt'` dataset, which consists of 3 columns: 2 feature columns and 1 class label column. The columns are separated via white spaces. If your implementation is correct, the last line should show a data array in below the code cell that has the following contents:

```
          x1        x2      y
0        -3.84    -4.40    0
1         16.36   6.54     1
2        -2.73    -5.13    0
3         4.83     7.22    1
4         3.66    -5.34    0
```

In [12]:

```
import pandas as pd

df_train = pd.read_csv('train_data.txt', delimiter='\s+')

df_train.head()
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[12], line 3
      1 import pandas as pd
----> 3 df_train = pd.read_csv('train_data.txt', delimiter='\s+')
      5 df_train.head()

File /usr/lib/python3/dist-packages/pandas/io/parsers/readers.py:948, in read_csv(filepath_or_buffer, sep, delimiter, header, names, index_col, usecols, dtype, engine, converters, true_values, false_values, skipinitialspace, skiprows, skipfooter, nrows, na_values, keep_default_na, na_filter, verbose, skip_blank_lines, parse_dates, infer_datetime_format, keep_date_col, date_parser, date_format, dayfirst, cache_dates, iterator, chunksize, compression, thousands, decimal, lineterminator, quotechar, quoting, doublequote, escapechar, comment, encoding, encoding_errors, dialect, on_bad_lines, delim_whitespace, low_memory, memory_map, float_precision, storage_options, dtype_backend)
    935 kwds_defaults = _refine_defaults_read(
    936     dialect,
    937     delimiter,
   (...)
    944     dtype_backend=dtype_backend,
    945 )
    946 kwds.update(kwds_defaults)
--> 948 return _read(filepath_or_buffer, kwds)

File /usr/lib/python3/dist-packages/pandas/io/parsers/readers.py:611, in _read(filepath_or_buffer, kwds)
    608 _validate_names(kwds.get("names", None))
    610 # Create the parser.
--> 611 parser = TextFileReader(filepath_or_buffer, **kwds)
    613 if chunksize or iterator:
    614     return parser

File /usr/lib/python3/dist-packages/pandas/io/parsers/readers.py:1448, in T
```

```
extFileReader.__init__(self, f, engine, **kwds)
   1445     self.options["has_index_names"] = kwds["has_index_names"]
   1447 self.handles: IOHandles | None = None
-> 1448 self._engine = self._make_engine(f, self.engine)

File /usr/lib/python3/dist-packages/pandas/io/parsers/readers.py:1705, in T
extFileReader._make_engine(self, f, engine)
   1703     if "b" not in mode:
   1704         mode += "b"
-> 1705 self.handles = get_handle(
   1706     f,
   1707     mode,
   1708     encoding=self.options.get("encoding", None),
   1709     compression=self.options.get("compression", None),
   1710     memory_map=self.options.get("memory_map", False),
   1711     is_text=is_text,
   1712     errors=self.options.get("encoding_errors", "strict"),
   1713     storage_options=self.options.get("storage_options", None),
   1714 )
   1715 assert self.handles is not None
   1716 f = self.handles.handle

File /usr/lib/python3/dist-packages/pandas/io/common.py:863, in get_handle
(path_or_buf, mode, encoding, compression, memory_map, is_text, errors, sto
rage_options)
    858 elif isinstance(handle, str):
    859     # Check whether the filename is to be opened in binary mode.
    860     # Binary mode does not support 'encoding' and 'newline'.
    861     if ioargs.encoding and "b" not in ioargs.mode:
    862         # Encoding
--> 863         handle = open(
    864             handle,
    865             ioargs.mode,
    866             encoding=ioargs.encoding,
    867             errors=errors,
    868             newline="",
    869         )
    870     else:
    871         # Binary mode
    872         handle = open(handle, ioargs.mode)

FileNotFoundError: [Errno 2] No such file or directory: 'train_data.txt'
```

E 14)

Consider the following code below, which plots one the samples from class 0 in a 2D scatterplot using matplotlib:

In [ ]:

```
X_train = df_train[['x1', 'x2']].values
y_train = df_train['y'].values
```

In [14]:

```
%matplotlib inline
import matplotlib.pyplot as plt
```

```python
# Scatter plot for class 0
plt.scatter(X_train[y_train == 0, 0],
            X_train[y_train == 0, 1],
            label='class 0', color='blue')

# Scatter plot for class 1
plt.scatter(X_train[y_train == 1, 0],
            X_train[y_train == 1, 1],
            label='class 1', color='red')

plt.xlabel('x1')
plt.ylabel('x2')
plt.xlim([-20, 20])
plt.ylim([-20, 20])
plt.legend(loc='upper left')
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[14], line 5
      2 import matplotlib.pyplot as plt
      4 # Scatter plot for class 0
----> 5 plt.scatter(X_train[y_train == 0, 0],
      6             X_train[y_train == 0, 1],
      7             label='class 0', color='blue')
      9 # Scatter plot for class 1
     10 plt.scatter(X_train[y_train == 1, 0],
     11             X_train[y_train == 1, 1],
     12             label='class 1', color='red')

NameError: name 'X_train' is not defined
```

Now, the following code below is identical to the code in the previous code cell but contains partial code to also include the examples from the second class. Complete the second `plt.scatter` function to also plot the trainign examples from `class 1`.

In [15]:

```python
plt.scatter(X_train[y_train == 0, 0],
            X_train[y_train == 0, 1],
            label='class 0', color='blue')

plt.scatter(X_train[y_train == 1, 0],
            X_train[y_train == 1, 1],
            label='class 1', color='red')

plt.xlabel('x1')
plt.ylabel('x2')
plt.xlim([-20, 20])
plt.ylim([-20, 20])
plt.legend(loc='upper left')
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[15], line 1
----> 1 plt.scatter(X_train[y_train == 0, 0],
      2             X_train[y_train == 0, 1],
      3             label='class 0', color='blue')
      5 plt.scatter(X_train[y_train == 1, 0],
      6             X_train[y_train == 1, 1],
      7             label='class 1', color='red')
      9 plt.xlabel('x1')

NameError: name 'X_train' is not defined
```

E 15)

Consider the we trained a 1-nearest neighbor classifier using scikit-learn on the previous training dataset:

In [17]:

```python
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

df_train = pd.read_csv('train_data.txt', delim_whitespace=True)

X_train = df_train[['x1', 'x2']].values
y_train = df_train['y'].values

knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)

y_pred = knn.predict(X_train)

accuracy = accuracy_score(y_train, y_pred)
print(f'Accuracy on training data: {accuracy:.2f}')
```

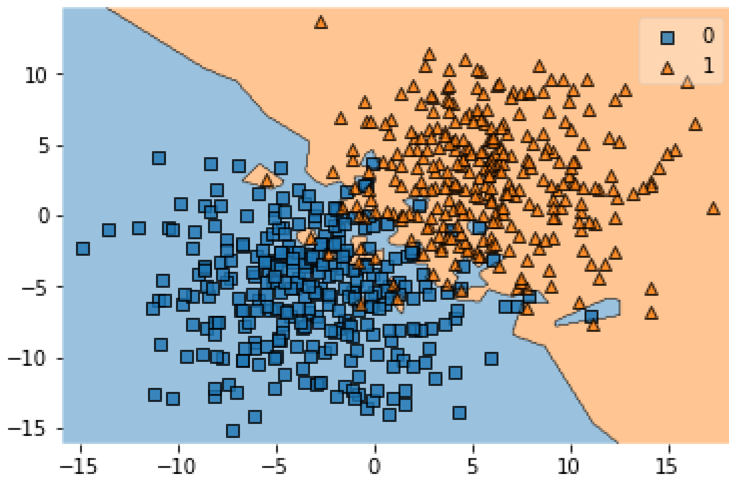
```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[17], line 6
      3 from sklearn.metrics import accuracy_score
      5 # Load the dataset
----> 6 df_train = pd.read_csv('train_data.txt', delim_whitespace=True)
      8 # Extract features and labels
      9 X_train = df_train[['x1', 'x2']].values

File /usr/lib/python3/dist-packages/pandas/io/parsers/readers.py:948, in re
ad_csv(filepath_or_buffer, sep, delimiter, header, names, index_col, usecol
s, dtype, engine, converters, true_values, false_values, skipinitialspace,
skiprows, skipfooter, nrows, na_values, keep_default_na, na_filter, verbos
e, skip_blank_lines, parse_dates, infer_datetime_format, keep_date_col, dat
e_parser, date_format, dayfirst, cache_dates, iterator, chunksize, compress
ion, thousands, decimal, lineterminator, quotechar, quoting, doublequote, e
scapechar, comment, encoding, encoding_errors, dialect, on_bad_lines, delim
_whitespace, low_memory, memory_map, float_precision, storage_options, dtyp
```

```
e_backend)
    935 kwds_defaults = _refine_defaults_read(
    936     dialect,
    937     delimiter,
   (...)
    944     dtype_backend=dtype_backend,
    945 )
    946 kwds.update(kwds_defaults)
--> 948 return _read(filepath_or_buffer, kwds)

File /usr/lib/python3/dist-packages/pandas/io/parsers/readers.py:611, in _r
ead(filepath_or_buffer, kwds)
    608 _validate_names(kwds.get("names", None))
    610 # Create the parser.
--> 611 parser = TextFileReader(filepath_or_buffer, **kwds)
    613 if chunksize or iterator:
    614     return parser

File /usr/lib/python3/dist-packages/pandas/io/parsers/readers.py:1448, in T
extFileReader.__init__(self, f, engine, **kwds)
   1445     self.options["has_index_names"] = kwds["has_index_names"]
   1447 self.handles: IOHandles | None = None
-> 1448 self._engine = self._make_engine(f, self.engine)

File /usr/lib/python3/dist-packages/pandas/io/parsers/readers.py:1705, in T
extFileReader._make_engine(self, f, engine)
   1703     if "b" not in mode:
   1704         mode += "b"
-> 1705 self.handles = get_handle(
   1706     f,
   1707     mode,
   1708     encoding=self.options.get("encoding", None),
   1709     compression=self.options.get("compression", None),
   1710     memory_map=self.options.get("memory_map", False),
   1711     is_text=is_text,
   1712     errors=self.options.get("encoding_errors", "strict"),
   1713     storage_options=self.options.get("storage_options", None),
   1714 )
   1715 assert self.handles is not None
   1716 f = self.handles.handle

File /usr/lib/python3/dist-packages/pandas/io/common.py:863, in get_handle
(path_or_buf, mode, encoding, compression, memory_map, is_text, errors, sto
rage_options)
    858 elif isinstance(handle, str):
    859     # Check whether the filename is to be opened in binary mode.
    860     # Binary mode does not support 'encoding' and 'newline'.
    861     if ioargs.encoding and "b" not in ioargs.mode:
    862         # Encoding
--> 863         handle = open(
    864             handle,
    865             ioargs.mode,
    866             encoding=ioargs.encoding,
    867             errors=errors,
    868             newline="",
    869         )
```

```
    870     else:
    871         # Binary mode
    872         handle = open(handle, ioargs.mode)

FileNotFoundError: [Errno 2] No such file or directory: 'train_data.txt'
```

In [ ]:
```python
from mlxtend.plotting import plot_decision_regions

plot_decision_regions(X_train, y_train, knn)
```

Out[ ]: `<matplotlib.axes._subplots.AxesSubplot at 0x1a1534b5c0>`



Compute the misclassification error of the 1-NN classifier on the training set:

In [ ]:
```python
# ENTER YOUR CODE HERE
```

E 16)

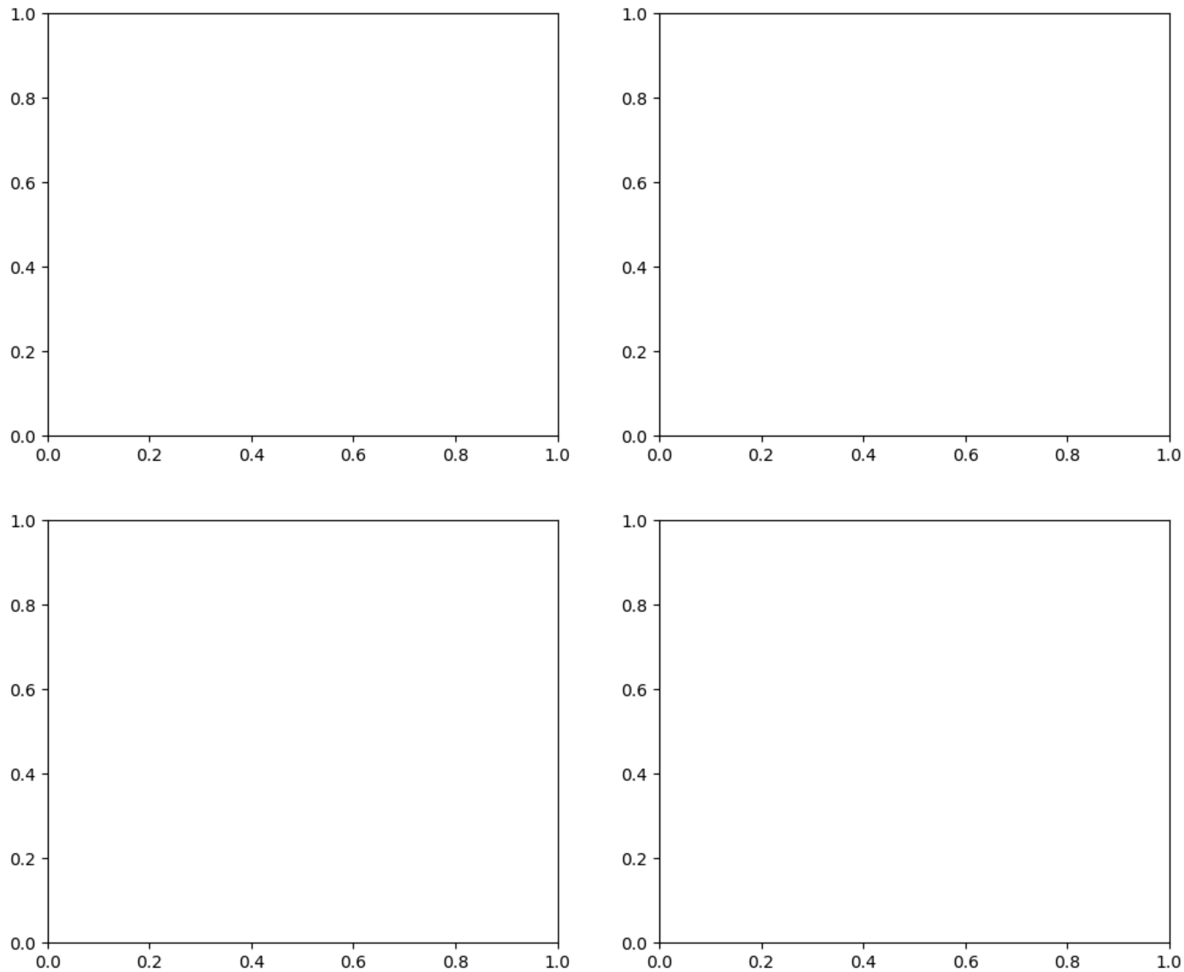
Use the code from E 15) to

- also visualize the decision boundaries of *k*-nearest neighbor classifiers with k=3, k=5, k=7, k=9
- compute the prediction error on the training set for the *k*-nearest neighbor classifiers with k=3, k=5, k=7, k=9

In [34]:
```python
# ENTER YOUR CODE HERE
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import make_classification

# Function to plot decision boundaries
def plot_decision_boundaries(X, y, k, ax):
    # Create an instance of the KNeighborsClassifier with k neighbors
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X, y)
```

```python
    # Create a mesh grid for plotting decision boundaries
    h = .02
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))

    Z = knn.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    ax.contourf(xx, yy, Z, alpha=0.3)
    ax.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k', marker='o', s=20)
    ax.set_title(f'k = {k}')

# Plotting
fig, axs = plt.subplots(2, 2, figsize=(12, 10))
plot_decision_boundaries(X_train_sub, y_train_sub, k=3, ax=axs[0, 0])
plot_decision_boundaries(X_train_sub, y_train_sub, k=5, ax=axs[0, 1])
plot_decision_boundaries(X_train_sub, y_train_sub, k=7, ax=axs[1, 0])
plot_decision_boundaries(X_train_sub, y_train_sub, k=9, ax=axs[1, 1])

for ax in axs.flat:
    ax.set_xticks([])
    ax.set_yticks([])

plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[34], line 29
     27 # Plotting
     28 fig, axs = plt.subplots(2, 2, figsize=(12, 10))
---> 29 plot_decision_boundaries(X_train_sub, y_train_sub, k=3, ax=axs[0,
0])
     30 plot_decision_boundaries(X_train_sub, y_train_sub, k=5, ax=axs[0,
1])
     31 plot_decision_boundaries(X_train_sub, y_train_sub, k=7, ax=axs[1,
0])

NameError: name 'X_train_sub' is not defined
```



In [ ]:

```
from sklearn.metrics import accuracy_score

# Function to compute and print accuracy for different k values
def compute_accuracy(k_values, X_train, y_train):
    for k in k_values:
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(X_train, y_train)
        y_train_pred = knn.predict(X_train)
        accuracy = accuracy_score(y_train, y_train_pred)
        print(f'Accuracy on training data with k={k}: {accuracy:.2f}')

# Compute accuracy for k=3, k=5, k=7, k=9
compute_accuracy(k_values=[3, 5, 7, 9], X_train=X_train_sub, y_train=y_train_sub)
```

In [ ]:

```
from sklearn.metrics import accuracy_score

# Function to compute and print accuracy for different k values
def compute_accuracy(k_values, X_train, y_train):
    for k in k_values:
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(X_train, y_train)
        y_train_pred = knn.predict(X_train)
        accuracy = accuracy_score(y_train, y_train_pred)
        print(f'Accuracy on training data with k={k}: {accuracy:.2f}')
```

```
# Compute accuracy for k=3, k=5, k=7, k=9
compute_accuracy(k_values=[3, 5, 7, 9], X_train=X_train_sub, y_train=y_train_sub)
```

In [ ]:
```
# ENTER YOUR CODE HERE
```

E 17)

Using the same approach you used in E 13), now load the `test_data.txt` file into a pandas array.

In [30]:
```
import pandas as pd

# Load the test dataset
df_test = pd.read_csv('test_data.txt', delim_whitespace=True)

# Display the first few rows of the DataFrame
df_test.head()
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[30], line 4
      1 import pandas as pd
      3 # Load the test dataset
----> 4 df_test = pd.read_csv('test_data.txt', delim_whitespace=True)
      6 # Display the first few rows of the DataFrame
      7 df_test.head()

File /usr/lib/python3/dist-packages/pandas/io/parsers/readers.py:948, in read_csv(filepath_or_buffer, sep, delimiter, header, names, index_col, usecols, dtype, engine, converters, true_values, false_values, skipinitialspace, skiprows, skipfooter, nrows, na_values, keep_default_na, na_filter, verbose, skip_blank_lines, parse_dates, infer_datetime_format, keep_date_col, date_parser, date_format, dayfirst, cache_dates, iterator, chunksize, compression, thousands, decimal, lineterminator, quotechar, quoting, doublequote, escapechar, comment, encoding, encoding_errors, dialect, on_bad_lines, delim_whitespace, low_memory, memory_map, float_precision, storage_options, dtype_backend)
    935 kwds_defaults = _refine_defaults_read(
    936     dialect,
    937     delimiter,
   (...)
    944     dtype_backend=dtype_backend,
    945 )
    946 kwds.update(kwds_defaults)
--> 948 return _read(filepath_or_buffer, kwds)

File /usr/lib/python3/dist-packages/pandas/io/parsers/readers.py:611, in _read(filepath_or_buffer, kwds)
    608 _validate_names(kwds.get("names", None))
    610 # Create the parser.
--> 611 parser = TextFileReader(filepath_or_buffer, **kwds)
    613 if chunksize or iterator:
```

```
    614     return parser

File /usr/lib/python3/dist-packages/pandas/io/parsers/readers.py:1448, in TextFileReader.__init__(self, f, engine, **kwds)
   1445     self.options["has_index_names"] = kwds["has_index_names"]
   1447 self.handles: IOHandles | None = None
-> 1448 self._engine = self._make_engine(f, self.engine)

File /usr/lib/python3/dist-packages/pandas/io/parsers/readers.py:1705, in TextFileReader._make_engine(self, f, engine)
   1703     if "b" not in mode:
   1704         mode += "b"
-> 1705 self.handles = get_handle(
   1706     f,
   1707     mode,
   1708     encoding=self.options.get("encoding", None),
   1709     compression=self.options.get("compression", None),
   1710     memory_map=self.options.get("memory_map", False),
   1711     is_text=is_text,
   1712     errors=self.options.get("encoding_errors", "strict"),
   1713     storage_options=self.options.get("storage_options", None),
   1714 )
   1715 assert self.handles is not None
   1716 f = self.handles.handle

File /usr/lib/python3/dist-packages/pandas/io/common.py:863, in get_handle(path_or_buf, mode, encoding, compression, memory_map, is_text, errors, storage_options)
    858 elif isinstance(handle, str):
    859     # Check whether the filename is to be opened in binary mode.
    860     # Binary mode does not support 'encoding' and 'newline'.
    861     if ioargs.encoding and "b" not in ioargs.mode:
    862         # Encoding
--> 863         handle = open(
    864             handle,
    865             ioargs.mode,
    866             encoding=ioargs.encoding,
    867             errors=errors,
    868             newline="",
    869         )
    870     else:
    871         # Binary mode
    872         handle = open(handle, ioargs.mode)

FileNotFoundError: [Errno 2] No such file or directory: 'test_data.txt'
```

Assign the features to `X_test` and the class labels to `y_test` (similar to E 13):

In [31]:
```python
# Assign features and class labels
X_test = df_test[['x1', 'x2']].values
y_test = df_test['y'].values
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[31], line 2
      1 # Assign features and class labels
```

```
----> 2 X_test = df_test[['x1', 'x2']].values
      3 y_test = df_test['y'].values

NameError: name 'df_test' is not defined
```

E 18)

Use the `train_test_split` function from scikit-learn to divide the training dataset further into a training subset and a validation set. The validation set should be 30% of the training dataset size, and the training subset should be 70% of the training dataset size.

For you reference, the `train_test_split` function is documented at http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html.

In [29]:
```
import pandas as pd
from sklearn.model_selection import train_test_split

# Load the dataset
df_train = pd.read_csv('train_data.txt', delim_whitespace=True)

# Extract features and labels
X_train = df_train[['x1', 'x2']].values
y_train = df_train['y'].values

# Perform the train-validation split
X_train_sub, X_val, y_train_sub, y_val = train_test_split(
    X_train, y_train,
    test_size=0.3,  # 30% of the training data will be used as validation data
    random_state=123,
    stratify=y_train  # Ensures the split maintains the same distribution of labels
)

print(f"Training subset shape: {X_train_sub.shape}")
print(f"Validation set shape: {X_val.shape}")
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[29], line 5
      2 from sklearn.model_selection import train_test_split
      4 # Load the dataset
----> 5 df_train = pd.read_csv('train_data.txt', delim_whitespace=True)
      7 # Extract features and labels
      8 X_train = df_train[['x1', 'x2']].values

File /usr/lib/python3/dist-packages/pandas/io/parsers/readers.py:948, in re
ad_csv(filepath_or_buffer, sep, delimiter, header, names, index_col, usecol
s, dtype, engine, converters, true_values, false_values, skipinitialspace,
skiprows, skipfooter, nrows, na_values, keep_default_na, na_filter, verbos
e, skip_blank_lines, parse_dates, infer_datetime_format, keep_date_col, dat
e_parser, date_format, dayfirst, cache_dates, iterator, chunksize, compress
ion, thousands, decimal, lineterminator, quotechar, quoting, doublequote, e
scapechar, comment, encoding, encoding_errors, dialect, on_bad_lines, delim
```

```
_whitespace, low_memory, memory_map, float_precision, storage_options, dtyp
e_backend)
    935 kwds_defaults = _refine_defaults_read(
    936     dialect,
    937     delimiter,
   (...)
    944     dtype_backend=dtype_backend,
    945 )
    946 kwds.update(kwds_defaults)
--> 948 return _read(filepath_or_buffer, kwds)

File /usr/lib/python3/dist-packages/pandas/io/parsers/readers.py:611, in _r
ead(filepath_or_buffer, kwds)
    608 _validate_names(kwds.get("names", None))
    610 # Create the parser.
--> 611 parser = TextFileReader(filepath_or_buffer, **kwds)
    613 if chunksize or iterator:
    614     return parser

File /usr/lib/python3/dist-packages/pandas/io/parsers/readers.py:1448, in T
extFileReader.__init__(self, f, engine, **kwds)
   1445     self.options["has_index_names"] = kwds["has_index_names"]
   1447 self.handles: IOHandles | None = None
-> 1448 self._engine = self._make_engine(f, self.engine)

File /usr/lib/python3/dist-packages/pandas/io/parsers/readers.py:1705, in T
extFileReader._make_engine(self, f, engine)
   1703     if "b" not in mode:
   1704         mode += "b"
-> 1705 self.handles = get_handle(
   1706     f,
   1707     mode,
   1708     encoding=self.options.get("encoding", None),
   1709     compression=self.options.get("compression", None),
   1710     memory_map=self.options.get("memory_map", False),
   1711     is_text=is_text,
   1712     errors=self.options.get("encoding_errors", "strict"),
   1713     storage_options=self.options.get("storage_options", None),
   1714 )
   1715 assert self.handles is not None
   1716 f = self.handles.handle

File /usr/lib/python3/dist-packages/pandas/io/common.py:863, in get_handle
(path_or_buf, mode, encoding, compression, memory_map, is_text, errors, sto
rage_options)
    858 elif isinstance(handle, str):
    859     # Check whether the filename is to be opened in binary mode.
    860     # Binary mode does not support 'encoding' and 'newline'.
    861     if ioargs.encoding and "b" not in ioargs.mode:
    862         # Encoding
--> 863         handle = open(
    864             handle,
    865             ioargs.mode,
    866             encoding=ioargs.encoding,
    867             errors=errors,
    868             newline="",
```

```
            )
        else:
            # Binary mode
            handle = open(handle, ioargs.mode)

FileNotFoundError: [Errno 2] No such file or directory: 'train_data.txt'
```

E 19)

Write a for loop to evaluate different *k* nn models with k=1 to k=14. In particular, fit the `KNeighborsClassifier` on the training subset, then evaluate it on the training subset, validation subset, and test subset. Report the respective classification error or accuracy.

In [27]:

```python
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Assume X and y are your full feature matrix and labels
X = # your feature matrix
y = # your labels

# Split the dataset into training + validation and test sets
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.2, random_state=123)

# Further split the training + validation set into training and validation sets
X_train_sub, X_val, y_train_sub, y_val = train_test_split(X_train_val, y_train_val, test_size=0.25

# Loop to evaluate k-NN models with different k values
for k in range(1, 15):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_sub, y_train_sub)

    # Predict on the training, validation, and test subsets
    y_train_pred = knn.predict(X_train_sub)
    y_val_pred = knn.predict(X_val)
    y_test_pred = knn.predict(X_test)

    # Calculate accuracy for training, validation, and test subsets
    train_accuracy = accuracy_score(y_train_sub, y_train_pred)
    val_accuracy = accuracy_score(y_val, y_val_pred)
    test_accuracy = accuracy_score(y_test, y_test_pred)

    # Print the results for the current k
    print(f'k={k}:')
    print(f'  Training Accuracy: {train_accuracy:.2f}')
    print(f'  Validation Accuracy: {val_accuracy:.2f}')
    print(f'  Test Accuracy: {test_accuracy:.2f}\n')
```

```
  Cell In[27], line 6
    X = # your feature matrix
        ^
SyntaxError: invalid syntax
```

E 20)

Consider the following code cell, where I implemented *k*-nearest neighbor classification algorithm following the the scikit-learn API

In [ ]:
```python
import numpy as np


class KNNClassifier(object):
    def __init__(self, k, dist_fn=None):
        self.k = k
        if dist_fn is None:
            self.dist_fn = self._euclidean_dist

    def _euclidean_dist(self, a, b):
        dist = 0.
        for ele_i, ele_j in zip(a, b):
            dist += ((ele_i - ele_j)**2)
        dist = dist**0.5
        return dist

    def _find_nearest(self, x):
        dist_idx_pairs = []
        for j in range(self.dataset_.shape[0]):
            d = self.dist_fn(x, self.dataset_[j])
            dist_idx_pairs.append((d, j))

        sorted_dist_idx_pairs = sorted(dist_idx_pairs)

        return sorted_dist_idx_pairs

    def fit(self, X, y):
        self.dataset_ = X.copy()
        self.labels_ = y.copy()
        self.possible_labels_ = np.unique(y)

    def predict(self, X):
        predictions = np.zeros(X.shape[0], dtype=int)
        for i in range(X.shape[0]):
            k_nearest = self._find_nearest(X[i])[:self.k]
            indices = [entry[1] for entry in k_nearest]
            k_labels = self.labels_[indices]
            counts = np.bincount(k_labels,
                                 minlength=self.possible_labels_.shape[0])
            pred_label = np.argmax(counts)
```

```
            predictions[i] = pred_label
        return predictions
```

In [ ]:
```
five_test_inputs = X_train[:5]
five_test_labels = y_train[:5]

knn = KNNClassifier(k=1)
knn.fit(five_test_inputs, five_test_labels)
print('True labels:', five_test_labels)
print('Pred labels:', knn.predict(five_test_inputs))
```

Since this is a very simple implementation of *k*NN, it is relatively slow -- very slow compared to the scikit-learn implementation which uses data structures such as Ball-tree and KD-tree to find the nearest neighbors more efficiently, as discussed in the lecture.

While we won't implement advanced data structures in this class, there is already an obvious opportunity for improving the computational efficiency by replacing for-loops with vectorized NumPy code (as discussed in the lecture). In particular, consider the `_euclidean_dist` method in the `KNNClassifier` class above. Below, I have written is as a function (as opposed to a method), for simplicity:

In [ ]:
```
def euclidean_dist(a, b):
    dist = 0.
    for ele_i, ele_j in zip(a, b):
        dist += ((ele_i - ele_j)**2)
    dist = dist**0.5
    return dist
```

Your task is now to benchmark this function using the `%timeit` magic command that we talked about in class using two random vectors, `a` and `b` as function inputs:

In [ ]:
```
rng = np.random.RandomState(123)

a = rng.rand(100)
b = rng.rand(100)
```

In [24]:
```
import numpy as np

def euclidean_dist(a, b):
    dist = 0.
    for ele_i, ele_j in zip(a, b):
        dist += ((ele_i - ele_j)**2)
    dist = dist**0.5
    return dist

rng = np.random.RandomState(123)
a = rng.rand(100)
b = rng.rand(100)

%timeit euclidean_dist(a, b)
```

```
30.8 µs ± 258 ns per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

E 21)

Now, rewrite the Euclidean distance function from E 20) in NumPy using

- either using the `np.sqrt` and `np.sum` function
- or using the `np.linalg.norm` function

and benchmark it again using the `%timeit` magic command. Then, compare results with the results you got in E 22). Did you make the function faster? Yes or No? Explain why, in 1-2 sentences.

In [22]:
```
# ENTER YOUR CODE HERE
import numpy as np

def euclidean_distance_np1(x, y):
    return np.sqrt(np.sum((x - y) ** 2))

# Benchmarking with %timeit
%timeit euclidean_distance_np1(np.random.rand(1000), np.random.rand(1000))
```

```
41.5 µs ± 615 ns per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

E 22)

Another inefficient aspect of the `KNNClassifier` implementation is that it uses the sorted function to sort all values in the distance value array. Since we are only interested in the *k* nearest neighbors, sorting *all* neighbors is quite unnecessary.

Consider the array `c`:

In [ ]:
```
rng = np.random.RandomState(123)
c = rng.rand(10000)
```

Call the sorted function to select the 3 smallest values in that array, we can do the following:

In [ ]:
```
sorted(c)[:3]
```

In the code cell below, use the `%timeit` magic command to benchmark the sorted command above:

In [18]:
```
# ENTER YOUR CODE HERE
import numpy as np

# Generate the array
rng = np.random.RandomState(123)
c = rng.rand(10000)
```

```
# Use %timeit to benchmark the sorting and selection
%timeit sorted(c)[:3]
```

```
3.13 ms ± 23.4 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

A more efficient way to select the *k* smallest values from an array is to use a priority queue, for example, implemented using a heap data structure. A convenient `nsmallest` function that does exactly that is available from Python's standard library:

In [ ]:
```
from heapq import nsmallest

nsmallest(3, c)
```

In the code cell below, use the `%timeit` magic command to benchmark the `nsmallest` function:

In [35]:
```
import numpy as np
from heapq import nsmallest

# Generate random array
rng = np.random.RandomState(123)
c = rng.rand(10000)

# Benchmark nsmallest using %timeit
%timeit nsmallest(3, c)
```

```
785 µs ± 18.2 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

Using nsmallest from the heapq module is more efficient than sorting the entire array when selecting the smallest k values. The nsmallest function performs better because it uses a heap-based approach that only focuses on the k smallest values, avoiding the overhead of sorting the entire dataset.

In [36]:
```
import numpy as np
from heapq import nsmallest
import time

# Generate random array
rng = np.random.RandomState(123)
c = rng.rand(10000)

# Benchmark nsmallest function
start_time = time.time()
smallest_values = nsmallest(3, c)
end_time = time.time()

print("Smallest values using nsmallest:", smallest_values)
print("Time taken with nsmallest:", end_time - start_time)
```

```
Smallest values using nsmallest: [6.783831227508141e-05, 8.188761366767494e
-05, 0.0001201014889748997]
```

```
Time taken with nsmallest: 0.0008616447448730469
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: