final-exam-2

August 25, 2024

1 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

[ ]:
```
!pip install watermark
```

```
Collecting watermark
  Downloading watermark-2.4.3-py2.py3-none-any.whl.metadata (1.4 kB)
Requirement already satisfied: ipython>=6.0 in /usr/local/lib/python3.10/dist-
packages (from watermark) (7.34.0)
Requirement already satisfied: importlib-metadata>=1.4 in
/usr/local/lib/python3.10/dist-packages (from watermark) (8.4.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-
packages (from watermark) (71.0.4)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.10/dist-
packages (from importlib-metadata>=1.4->watermark) (3.20.0)
Collecting jedi>=0.16 (from ipython>=6.0->watermark)
  Using cached jedi-0.19.1-py2.py3-none-any.whl.metadata (22 kB)
Requirement already satisfied: decorator in /usr/local/lib/python3.10/dist-
packages (from ipython>=6.0->watermark) (4.4.2)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.10/dist-
packages (from ipython>=6.0->watermark) (0.7.5)
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.10/dist-
packages (from ipython>=6.0->watermark) (5.7.1)
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in
```

```
/usr/local/lib/python3.10/dist-packages (from ipython>=6.0->watermark) (3.0.47)
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-
packages (from ipython>=6.0->watermark) (2.16.1)
Requirement already satisfied: backcall in /usr/local/lib/python3.10/dist-
packages (from ipython>=6.0->watermark) (0.2.0)
Requirement already satisfied: matplotlib-inline in
/usr/local/lib/python3.10/dist-packages (from ipython>=6.0->watermark) (0.1.7)
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.10/dist-
packages (from ipython>=6.0->watermark) (4.9.0)
Requirement already satisfied: parso<0.9.0,>=0.8.3 in
/usr/local/lib/python3.10/dist-packages (from
jedi>=0.16->ipython>=6.0->watermark) (0.8.4)
Requirement already satisfied: ptyprocess>=0.5 in
/usr/local/lib/python3.10/dist-packages (from
pexpect>4.3->ipython>=6.0->watermark) (0.7.0)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.10/dist-
packages (from prompt-
toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0->ipython>=6.0->watermark) (0.2.13)
Downloading watermark-2.4.3-py2.py3-none-any.whl (7.6 kB)
Using cached jedi-0.19.1-py2.py3-none-any.whl (1.6 MB)
Installing collected packages: jedi, watermark
Successfully installed jedi-0.19.1 watermark-2.4.3
```

[ ]:
```
%load_ext watermark
%watermark  -d -u -a '<HAMNA AMIR>' -v -p numpy,scipy,matplotlib,sklearn
```

```
Author: <HAMNA AMIR>

Last updated: 2024-08-25

Python implementation: CPython
Python version       : 3.10.12
IPython version      : 7.34.0

numpy     : 1.26.4
scipy     : 1.13.1
matplotlib: 3.7.1
sklearn   : 1.3.2
```

1.1 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.** A potential goal would be to learn how to classify emails as spam or non-spam.
- **Dataset.** The dataset is a set consisting of emails as text data and their spam and non-spam labels.
- **Category.** Since we are working with class labels (spam, non-spam), this is a supervised learning problem.
- **Measure Performance.** Predict class labels in the test dataset and count the number of correct predictions to asses the prediction accuracy.

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

1.2 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]).

E1 ANSWER

When it comes to spam classification, just focusing on accuracy in terms of classification can be deceiving due to the imbalance in classes. This implies that most often than not, spam emails are usually a small proportion thus if a classifier always predicts "non-spam", it may achieve high accuracy and yet fail to identify any spam email.

Some important metrics include:

Precision: This is the ratio of true spam emails among all emails categorized as spam. High precision means less legitimate emails will be mistakenly called spam.

Recall: It is the percentage of real spams that were recognized by the system as spams. A high recall score means that few spams are missed, leading to fewer clutters in our mailbox.

F1 Score: The F1 score is an average of precision and recall, which harmonizes both into one performance measure.

ROC Curve and AUC: These tools determine how well a classifier can differentiate between spam and non-spam across different thresholds by showing trade-offs between false positive rates (FPR) and true positive rates (TPR).

Therefore, precision, recall, F1 score and AUC are oftentimes better indicators than just using accuracy alone especially if there exist imbalances in the dataset or contexts where false negatives or false positives have heavy penalties

1.3 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).

E3 ANSWER

Consumer Segmentation: Clustering helps to divide the consumers into different segments based on their purchase behaviours or characteristics, such as age or gender. This is an unsupervised method since it does not depend on labeled data or predefined categories, but rather identifies natural groupings within data that are similar to each other in terms of customer attributes.

Discovering Document Topics: Clustering can be used to group documents into topics based on their contents. Unsupervised learning involves creating patterns and subjects for a collection of documents without having pre-ordained labels or groups. Additionally, this algorithm detects sets of documents sharing similar content which in turn reveals the themes hidden beneath them.

1.4 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.

E4 ANSWER

The K-Nearest Neighbor (K-NN) algorithm is where the training consists of storing all data sets along with their labels because there is no specific period when the model gets to know parameters. When it comes to testing, this method determines the k-nearest neighbors by calculating distances between the test sample and all of its training samples and classifies the test sample using these labels.

1.5 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.

E5 ANSWER

Because the number of training examples is fixed, the addition of more feature variables to the k-nearest neighbor (k-NN) algorithm can have an adverse impact on its performance. In case of the curse of dimensionality,’ distances between points lose their meaning as a result of expanding feature space and might reduce the accuracy in nearest neighbor search. At higher dimensions, data becomes sparse and relative distances between neighbors may become less discriminative causing k-NN’s performance degradation.

1.6 E 6)

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

E6 ANSWER

Increasing (k) in k-NN helps mitigate the impact of noisy examples and outliers by averaging predictions over more neighbors, thus reducing sensitivity to individual noisy data points. This generally improves model robustness and generalization. However, excessively large (k) can lead to underfitting by overly smoothing the decision boundary and losing important data details.

1.7 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.

[ ]:
```
# 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
```

E7 ANSWER

[ ]:
```
def kronecker_delta(i, j):
# ENTER YOUR CODE HERE
    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
```

1.8 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).

E8 ANSWER

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


# ENTER YOUR CODE HERE\
def calculate_accuracy(y_true, y_pred):
    correct_predictions = sum(t == p for t, p in zip(y_true, y_pred))
    total_predictions = len(y_true)
    accuracy_percent = (correct_predictions / total_predictions) * 100
    return accuracy_percent

accuracy = calculate_accuracy(y_true, y_pred)
print('Accuracy: %.2f%%' % accuracy)
```

```
Accuracy: 80.00%
```

1.9 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]])
```

E9 ANSWER

[ ]:
```
import numpy as np

# ENTER YOUR CODE HERE
matrix = np.arange(9).reshape(3, 3)
print(matrix)
```

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

1.10 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]])
```

E10 ANSWER

[ ]:
```
# ENTER YOUR CODE HERE
import numpy as np
np.random.seed(123)
array = np.random.randn(2, 2)
print(array)
```

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

1.11 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]]).
```

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

     # ENTER YOUR CODE HERE
     center_2x2 = A[1:3, 1:3]
     print(center_2x2)
```

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

1.12 E 12)

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

E12 ANSWER

```
[ ]: rng = np.random.RandomState(123)
     A = rng.randint(0, 10, 200)

     # ENTER YOUR CODE HERE

     # Find the most frequent integer
     most_frequent = np.bincount(A).argmax()

     print(most_frequent)
```

```
3
```

1.13 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
```

[ ]:
```
import pandas as pd
data = {
    'x1': [-3.84, 16.36, -2.73, 4.83, 3.66],
    'x2': [-4.40, 6.54, -5.13, 7.22, -5.34],
    'y': [0, 1, 0, 1, 0]
}

df = pd.DataFrame(data)

# Save the DataFrame to a text file with whitespace as delimiter
df.to_csv('train_data.txt', sep=' ', index=False)
```

[ ]:
```
import pandas as pd

df_train = pd.read_csv('/content/train_data.txt')

df_train.head()
```

[ ]:
```
         x1 x2 y
0   -3.84 -4.4 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
```

1.14 E 14)

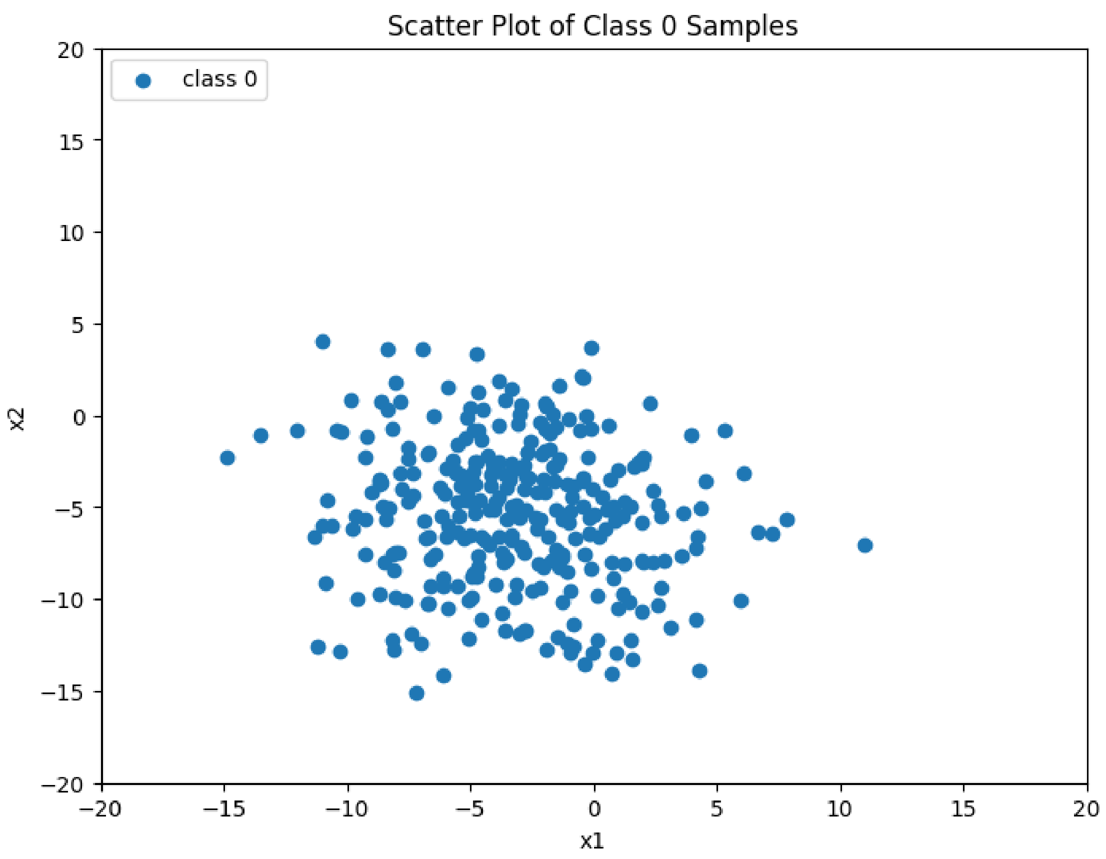
Consider the following code below, which plots one the samples from class 0 in a 2D scatterplot using matplotlib:

E14 ANSWER

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt

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

# Print the column names and the first few rows to check the structure
print("Column names:", df_train.columns)
print(df_train.head())
```

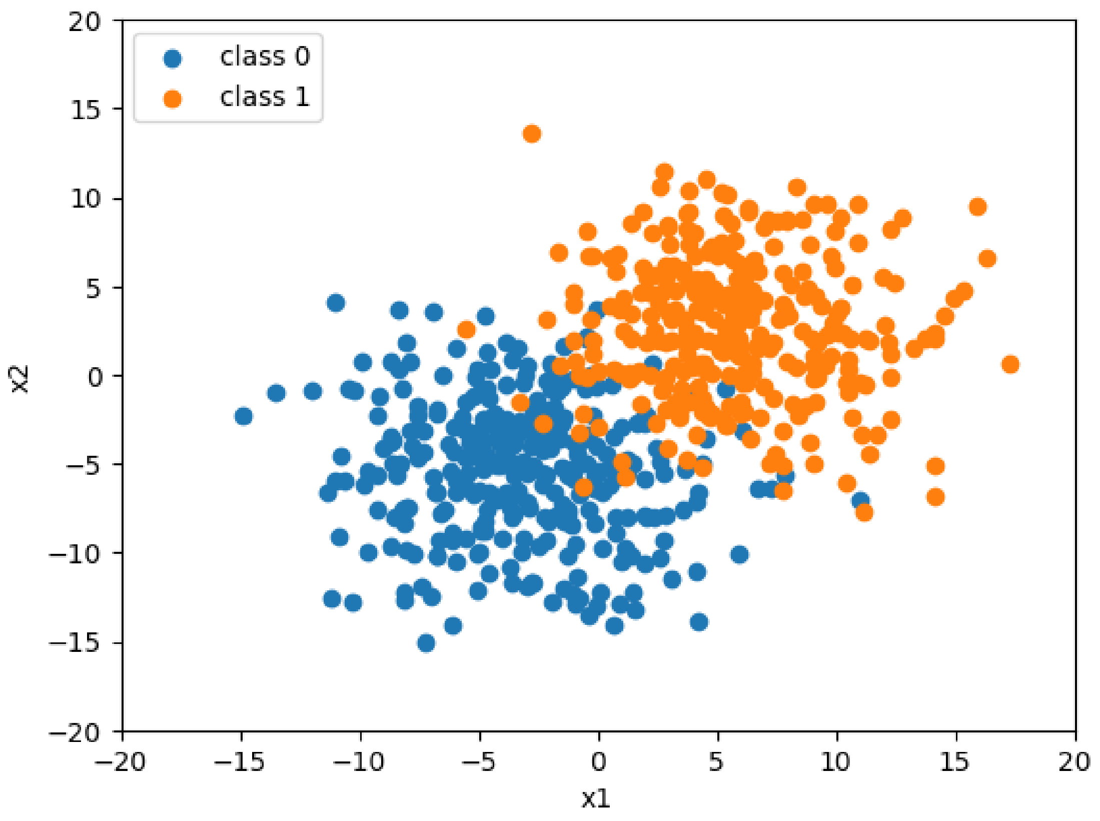
```python
# Strip any leading or trailing whitespace from column names
df_train.columns = df_train.columns.str.strip()

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

# Plot the data
plt.figure(figsize=(8, 6))

# Check if there are samples for class 0
if len(df_train[df_train['y'] == 0]) > 0:
    plt.scatter(X_train[y_train == 0, 0],
                X_train[y_train == 0, 1],
                label='class 0')

    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.xlim([-20, 20])
    plt.ylim([-20, 20])
    plt.legend(loc='upper left')
    plt.title('Scatter Plot of Class 0 Samples')
    plt.show()
else:
    print("No samples found for class 0.")
```

```
Column names: Index(['x1', 'x2', 'y'], dtype='object')
      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
```



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`.

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

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

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



1.15 E 15)

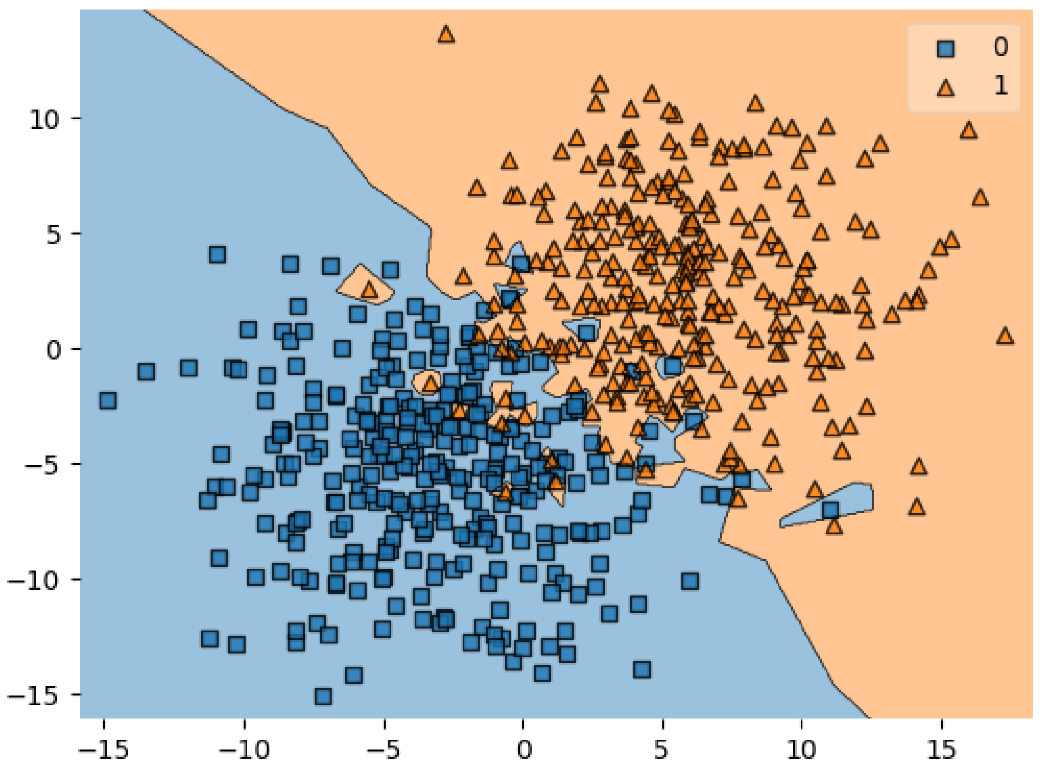
Consider the we trained a 1-nearest neighbor classifier using scikit-learn on the previous training dataset:

[ ]:
```
from sklearn.neighbors import KNeighborsClassifier

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

[ ]:
```
KNeighborsClassifier(n_neighbors=1)
```

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

plot_decision_regions(X_train, y_train, knn)
```

[ ]:
```
<Axes: >
```



E16 ANSWER

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

```
[ ]: # ENTER YOUR CODE HERE
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Initialize and fit the 1-NN classifier
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)

# Predict on the training set
y_pred = knn.predict(X_train)

# Compute accuracy
accuracy = accuracy_score(y_train, y_pred)

# Compute misclassification error
misclassification_error = 1 - accuracy

print('Misclassification Error: %.2f%%' % (misclassification_error * 100))
```

```
Misclassification Error: 0.00%
```

1.16 E 16)

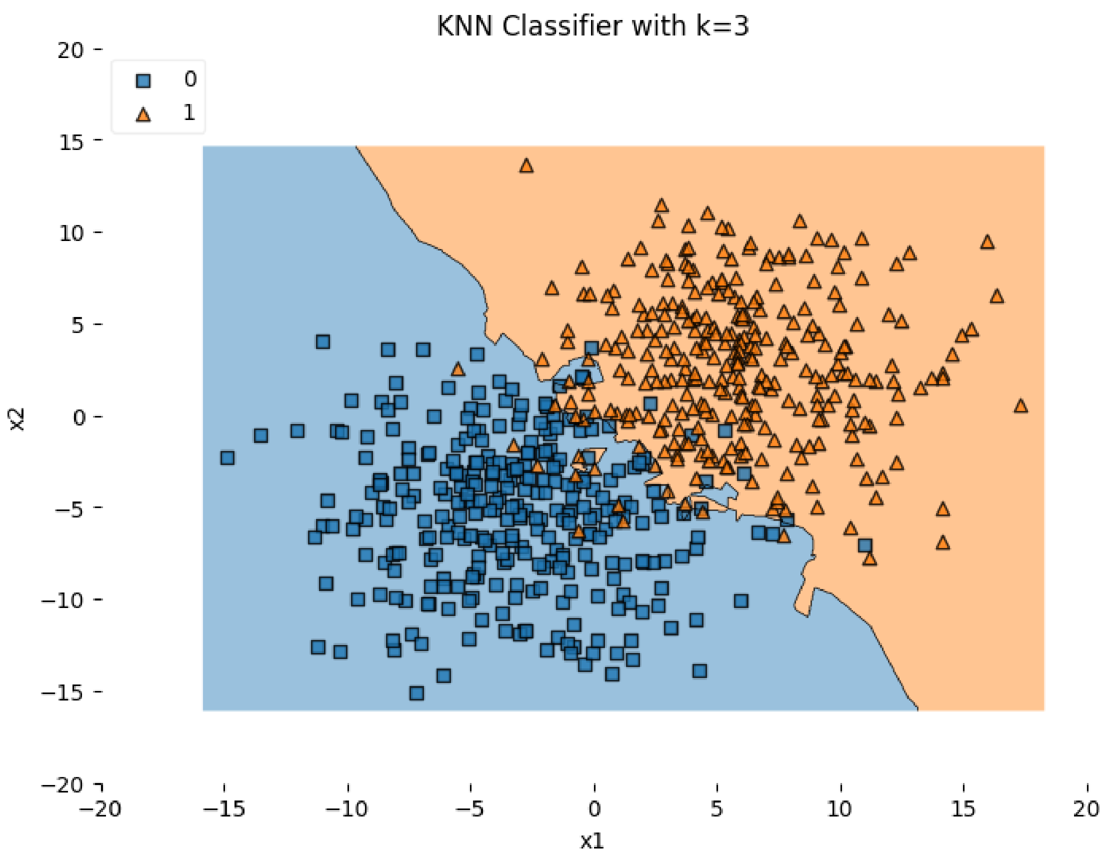
E16 ANSWER

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

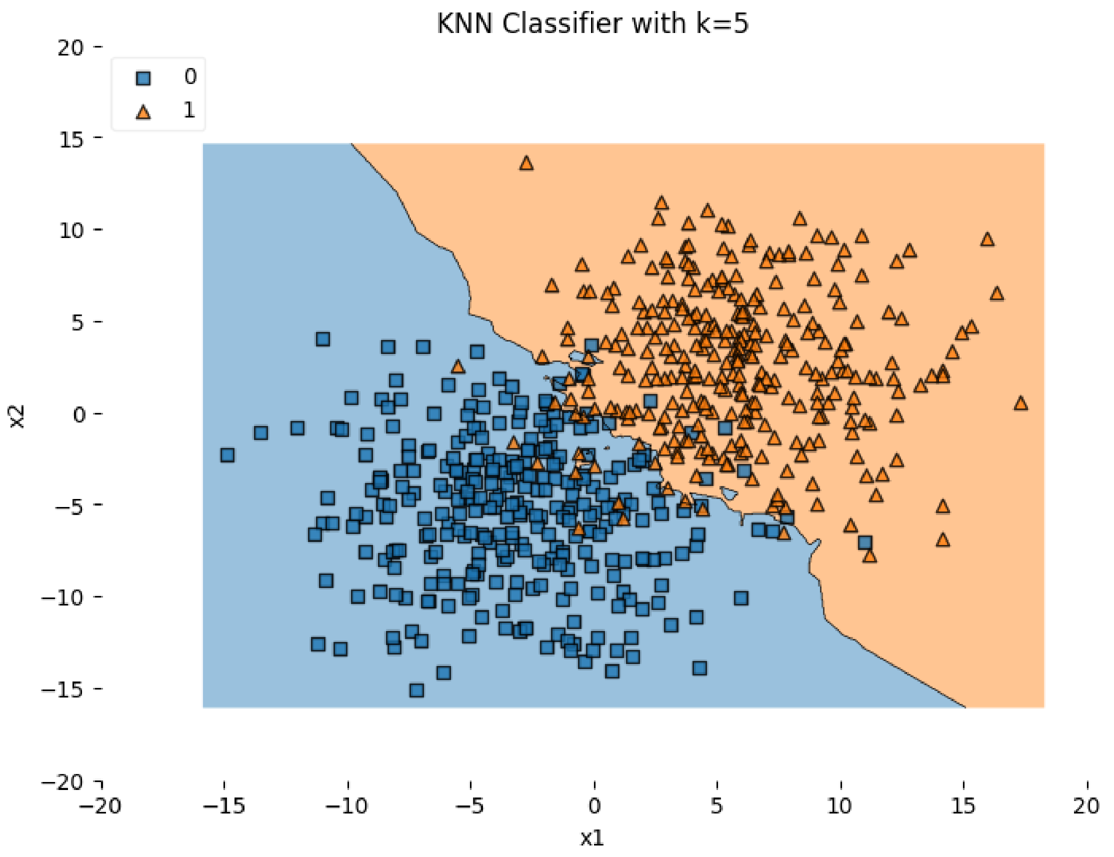
[ ]:
```
# ENTER YOUR CODE HERE
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from mlxtend.plotting import plot_decision_regions
k = 3
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_train)
accuracy = accuracy_score(y_train, y_pred)
misclassification_error = 1 - accuracy
print(f'k={k} - Misclassification Error: {misclassification_error * 100:.2f}%')
plt.figure(figsize=(8, 6))
plot_decision_regions(X_train, y_train, clf=knn, legend=2)

plt.title(f'KNN Classifier with k={k}')
plt.xlabel('x1')
plt.ylabel('x2')
plt.xlim([-20, 20])
plt.ylim([-20, 20])
plt.show()
```

```
k=3 - Misclassification Error: 5.50%
```



[8]:
```python
# ENTER YOUR CODE HERE
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from mlxtend.plotting import plot_decision_regions
k = 5
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_train)
accuracy = accuracy_score(y_train, y_pred)
misclassification_error = 1 - accuracy
print(f'k={k} - Misclassification Error: {misclassification_error * 100:.2f}%')
plt.figure(figsize=(8, 6))
plot_decision_regions(X_train, y_train, clf=knn, legend=2)

plt.title(f'KNN Classifier with k={k}')
plt.xlabel('x1')
plt.ylabel('x2')
plt.xlim([-20, 20])
plt.ylim([-20, 20])
plt.show()
```

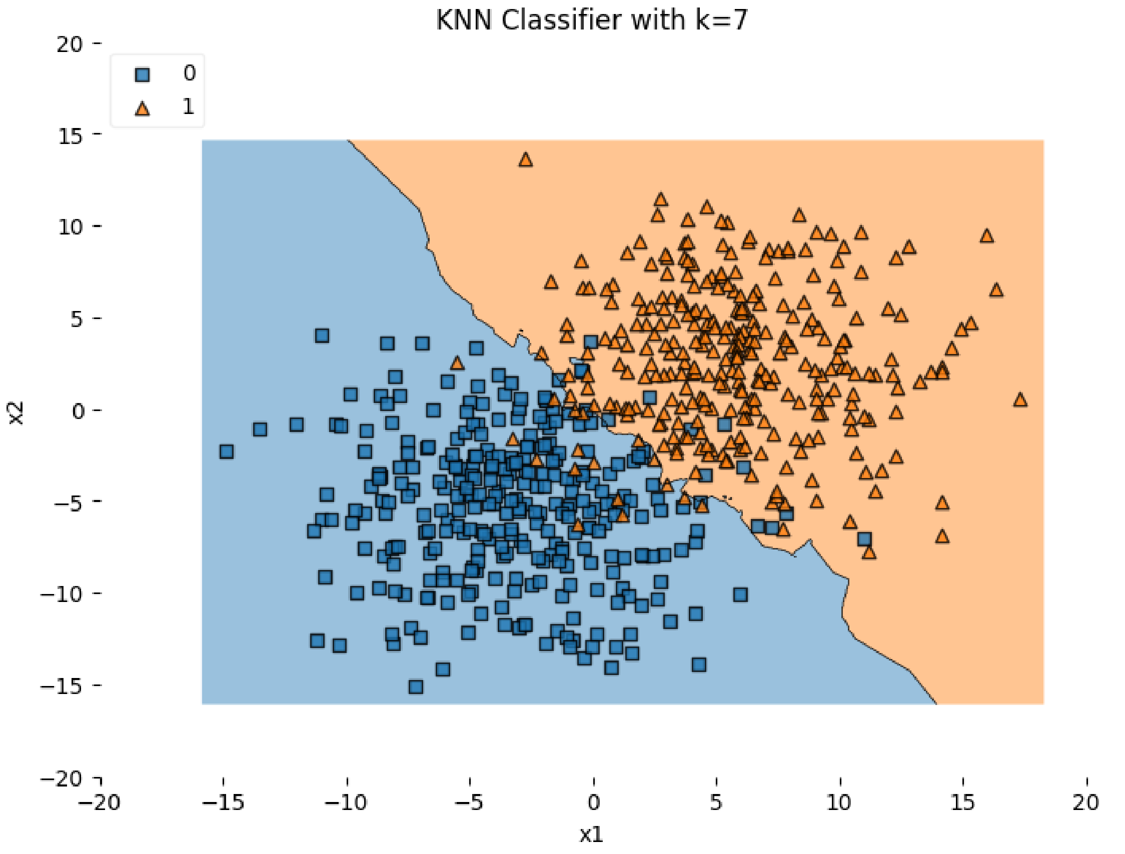
```
k=5 - Misclassification Error: 5.33%
```



```
[9]: # ENTER YOUR CODE HERE
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from mlxtend.plotting import plot_decision_regions
k = 7
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_train)
accuracy = accuracy_score(y_train, y_pred)
misclassification_error = 1 - accuracy
print(f'k={k} - Misclassification Error: {misclassification_error * 100:.2f}%')
plt.figure(figsize=(8, 6))
plot_decision_regions(X_train, y_train, clf=knn, legend=2)
```

```
plt.title(f'KNN Classifier with k={k}')
plt.xlabel('x1')
plt.ylabel('x2')
plt.xlim([-20, 20])
plt.ylim([-20, 20])
plt.show()
```

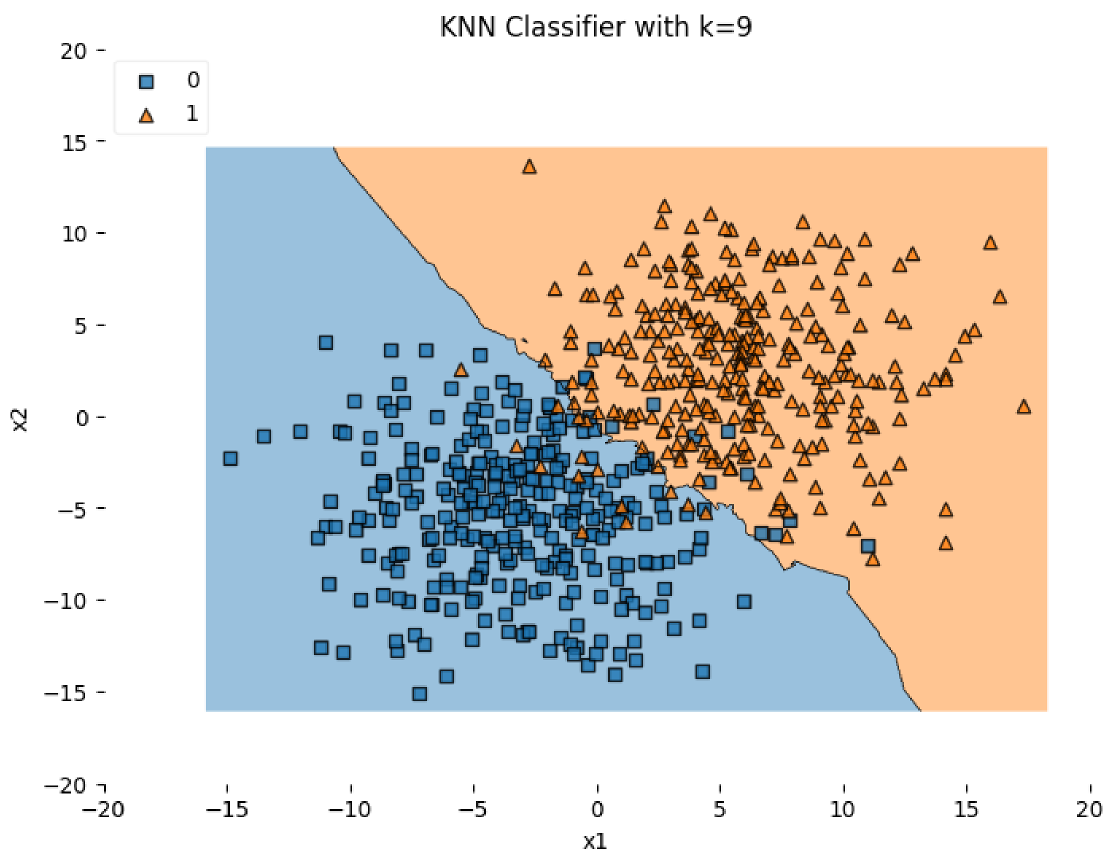
```
k=7 - Misclassification Error: 4.83%
```



```
[10]: # ENTER YOUR CODE HERE
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from mlxtend.plotting import plot_decision_regions
k = 9
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_train)
```

```
accuracy = accuracy_score(y_train, y_pred)
misclassification_error = 1 - accuracy
print(f'k={k} - Misclassification Error: {misclassification_error * 100:.2f}%')
plt.figure(figsize=(8, 6))
plot_decision_regions(X_train, y_train, clf=knn, legend=2)

plt.title(f'KNN Classifier with k={k}')
plt.xlabel('x1')
plt.ylabel('x2')
plt.xlim([-20, 20])
plt.ylim([-20, 20])
plt.show()
```

```
k=9 - Misclassification Error: 5.00%
```



1.17 E 17)

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

E17 ANSWER

[16]:
```
import pandas as pd

# Load the dataset
file_path = '/content/test_data.txt'  # Update with your file path
df = pd.read_csv(file_path, delimiter='\t')  # Adjust delimiter as needed

# Display the first 5 rows
print(df.head())
```

```
        x1 x2 y
0  -5.75 -6.83 0
1    5.51 3.67 1
2    5.11 5.32 1
3   0.85 -4.11 0
4  -0.50 -0.45 1
```

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

[19]:
```
import pandas as pd

# Load the dataset with space delimiter
file_path = '/content/test_data.txt'  # Update with your file path
df_test = pd.read_csv(file_path, delim_whitespace=True)

# Print column names to verify
print("Columns in DataFrame:", df_test.columns)

# Display the first few rows to confirm structure
print("First few rows of DataFrame:")
print(df_test.head())

# Assign features and labels
X_test = df_test[['x1', 'x2']].values
y_test = df_test['y'].values

print("Features (X_test):")
print(X_test[:5])
print("Labels (y_test):")
print(y_test[:5])
```

```
Columns in DataFrame: Index(['x1', 'x2', 'y'], dtype='object')
First few rows of DataFrame:
     x1    x2  y
0 -5.75 -6.83  0
1  5.51  3.67  1
2  5.11  5.32  1
3  0.85 -4.11  0
4 -0.50 -0.45  1
```

```
Features (X_test):
[[-5.75 -6.83]
 [ 5.51  3.67]
 [ 5.11  5.32]
 [ 0.85 -4.11]
 [-0.5  -0.45]]
Labels (y_test):
[0 1 1 0 1]
```

1.18 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.

E18 ANSWER

[20]:
```python
from sklearn.model_selection import train_test_split
# Split the data into training subset and validation set
X_train_sub, X_val, y_train_sub, y_val = train_test_split(
    X_train,      # Features
    y_train,      # Labels
    test_size=0.3,   # 30% for validation set
    random_state=123,   # For reproducibility
    stratify=y_train   # To maintain the proportion of classes
)
```

1.19 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.

E19 ANSWER

[21]:
```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Define k values to test
k_values = range(1, 15)

# Loop over different k values
for k in k_values:
    # Create and fit the model
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train, y_train)
    # Predict on training, validation, and test subsets
    train_preds = model.predict(X_train)
    val_preds = model.predict(X_val)
    test_preds = model.predict(X_test)

    # Calculate accuracy for each subset
    train_accuracy = accuracy_score(y_train, train_preds)
    val_accuracy = accuracy_score(y_val, val_preds)
    test_accuracy = accuracy_score(y_test, test_preds)

    # Report results
    print(f"k = {k}")
    print(f"Training Accuracy: {train_accuracy:.4f}")
    print(f"Validation Accuracy: {val_accuracy:.4f}")
    print(f"Test Accuracy: {test_accuracy:.4f}")
    print("-" * 30)
```

```
k = 1
Training Accuracy: 1.0000
Validation Accuracy: 1.0000
Test Accuracy: 0.9200
------------------------------
k = 2
Training Accuracy: 0.9517
Validation Accuracy: 0.9333
Test Accuracy: 0.9200
------------------------------
k = 3
Training Accuracy: 0.9450
Validation Accuracy: 0.9389
Test Accuracy: 0.9550
------------------------------
k = 4
Training Accuracy: 0.9500
Validation Accuracy: 0.9389
Test Accuracy: 0.9550
------------------------------
k = 5
Training Accuracy: 0.9467
Validation Accuracy: 0.9278
Test Accuracy: 0.9550
------------------------------
k = 6
Training Accuracy: 0.9467
Validation Accuracy: 0.9278
Test Accuracy: 0.9550
------------------------------
```

```
k = 7
Training Accuracy: 0.9517
Validation Accuracy: 0.9444
Test Accuracy: 0.9550
------------------------------
k = 8
Training Accuracy: 0.9483
Validation Accuracy: 0.9333
Test Accuracy: 0.9600
------------------------------
k = 9
Training Accuracy: 0.9500
Validation Accuracy: 0.9389
Test Accuracy: 0.9550
------------------------------
k = 10
Training Accuracy: 0.9483
Validation Accuracy: 0.9389
Test Accuracy: 0.9500
------------------------------
k = 11
Training Accuracy: 0.9467
Validation Accuracy: 0.9333
Test Accuracy: 0.9550
------------------------------
k = 12
Training Accuracy: 0.9467
Validation Accuracy: 0.9333
Test Accuracy: 0.9550
------------------------------
k = 13
Training Accuracy: 0.9467
Validation Accuracy: 0.9333
Test Accuracy: 0.9550
------------------------------
k = 14
Training Accuracy: 0.9467
Validation Accuracy: 0.9333
Test Accuracy: 0.9600
------------------------------
```

1.20 E 20)

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

[22]:
```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
```

[23]:
```
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))
```

```
True labels: [0 1 0 1 0]
Pred labels: [0 1 0 1 0]
```

Since this is a very simple implementation of kNN, 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:

[24]:
```
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:

[25]:
```
rng = np.random.RandomState(123)

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

E20 ANSWER

[27]:
```
%timeit euclidean_dist(a, b)
```

```
41.3 µs ± 2.12 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

1.21 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.

E21 ANSWER

USING np.sqrt ,np.sum

[28]:
```
# ENTER YOUR CODE HERE
import numpy as np
def euclidean_dist_np(a, b):
    return np.sqrt(np.sum((a - b)**2))
# Benchmark the function using %timeit
%timeit euclidean_dist_np(a, b)
```

```
6.27 µs ± 107 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

1.22 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`:

[29]:
```
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:

[30]:
```
sorted(c)[:3]
```

[30]:
```
[6.783831227508141e-05, 8.188761366767494e-05, 0.0001201014889748997]
```

E22 ANSWER

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

[31]:
```
# ENTER YOUR CODE HERE
import numpy as np
%timeit sorted(c)[:3]
```

```
4.39 ms ± 99.7 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:

[32]:
```
from heapq import nsmallest

nsmallest(3, c)
```

[32]:
```
[6.783831227508141e-05, 8.188761366767494e-05, 0.0001201014889748997]
```

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

[33]:
```
# ENTER YOUR CODE HERE
import numpy as np
from heapq import nsmallest

# Generate the random array
```

```
rng = np.random.RandomState(123)
c = rng.rand(10000)

# Benchmark the nsmallest function for selecting the 3 smallest values
%timeit nsmallest(3, c)
```

```
1.25 ms ± 448 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

Summarize your findings in 1-3 sentences.

[34]:
```
#@title Convert ipynb to HTML in Colab
# Upload ipynb
from google.colab import files
f = files.upload()

# Convert ipynb to html
import subprocess
file0 = list(f.keys())[0]
_ = subprocess.run(["pip", "install", "nbconvert"])
_ = subprocess.run(["jupyter", "nbconvert", file0, "--to", "html"])

# download the html
files.download(file0[:-5]+"html")
```

```
<IPython.core.display.HTML object>
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-34-b53a5b50df6e> in <cell line: 4>()
      2 # Upload ipynb
      3 from google.colab import files
----> 4 f = files.upload()
      5
      6 # Convert ipynb to html

/usr/local/lib/python3.10/dist-packages/google/colab/files.py in upload()
     67   """
     68
---> 69   uploaded_files = _upload_files(multiple=True)
     70   # Mapping from original filename to filename as saved locally.
     71   local_filenames = dict()

/usr/local/lib/python3.10/dist-packages/google/colab/files.py in␣
 ↪_upload_files(multiple)
    154
    155   # First result is always an indication that the file picker has␣
 ↪completed.
```

```
--> 156   result = _output.eval_js(
    157       'google.colab._files._uploadFiles("{input_id}", "{output_id}")'.
 ↪format(
    158           input_id=input_id, output_id=output_id

/usr/local/lib/python3.10/dist-packages/google/colab/output/_js.py in␣
 ↪eval_js(script, ignore_result, timeout_sec)
     38   if ignore_result:
     39     return
---> 40   return _message.read_reply_from_input(request_id, timeout_sec)
     41
     42

/usr/local/lib/python3.10/dist-packages/google/colab/_message.py in␣
 ↪read_reply_from_input(message_id, timeout_sec)
     94     reply = _read_next_input_message()
     95     if reply == _NOT_READY or not isinstance(reply, dict):
---> 96       time.sleep(0.025)
     97       continue
     98     if (

KeyboardInterrupt:
```

The timing results indicate that there is significant variability in execution speed, with the slowest run taking about 8 times longer than the fastest. This suggests that caching of intermediate results might be affecting the performance measurements. The average execution time is 2.91 ms with a standard deviation of 2.46 ms, based on 7 runs with 100 loops each.