final-exam

August 27, 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

[16]:
```
%load_ext watermark
%watermark  -d -u -a '<Your Name>' -v -p numpy,scipy,matplotlib,sklearn
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[16], line 1
----> 1 get_ipython().run_line_magic('load_ext', 'watermark')
      2 get_ipython().run_line_magic('watermark', " -d -u -a '<Your Name>' -v -p␣
 ↪numpy,scipy,matplotlib,sklearn")

File␣
 ↪~\AppData\Roaming\Python\Python312\site-packages\IPython\core\interactiveshel..
 ↪py:2480, in InteractiveShell.run_line_magic(self, magic_name, line,␣
 ↪_stack_depth)
   2478     kwargs['local_ns'] = self.get_local_scope(stack_depth)
   2479 with self.builtin_trap:
-> 2480     result = fn(*args, **kwargs)
   2482 # The code below prevents the output from being displayed
   2483 # when using magics with decorator @output_can_be_silenced
   2484 # when the last Python token in the expression is a ';'.
   2485 if getattr(fn, magic.MAGIC_OUTPUT_CAN_BE_SILENCED, False):
```

```
File␣
 ↪~\AppData\Roaming\Python\Python312\site-packages\IPython\core\magics\extensi
 ↪py:33, in ExtensionMagics.load_ext(self, module_str)
     31 if not module_str:
     32     raise UsageError('Missing module name.')
---> 33 res = self.shell.extension_manager.load_extension(module_str)
     35 if res == 'already loaded':
     36     print("The %s extension is already loaded. To reload it, use:" %␣
 ↪module_str)

File ~\AppData\Roaming\Python\Python312\site-packages\IPython\core\extensions.p
 ↪62, in ExtensionManager.load_extension(self, module_str)
     55 """Load an IPython extension by its module name.
     56
     57 Returns the string "already loaded" if the extension is already loaded,
     58 "no load function" if the module doesn't have a load_ipython_extension
     59 function, or None if it succeeded.
     60 """
     61 try:
---> 62     return self._load_extension(module_str)
     63 except ModuleNotFoundError:
     64     if module_str in BUILTINS_EXTS:

File ~\AppData\Roaming\Python\Python312\site-packages\IPython\core\extensions.p
 ↪77, in ExtensionManager._load_extension(self, module_str)
     75 with self.shell.builtin_trap:
     76     if module_str not in sys.modules:
---> 77         mod = import_module(module_str)
     78     mod = sys.modules[module_str]
     79     if self._call_load_ipython_extension(mod):

File e:\New folder (2)\Lib\importlib\__init__.py:90, in import_module(name,␣
 ↪package)
     88             break
     89         level += 1
---> 90 return _bootstrap._gcd_import(name[level:], package, level)

File <frozen importlib._bootstrap>:1387, in _gcd_import(name, package, level)

File <frozen importlib._bootstrap>:1360, in _find_and_load(name, import_)

File <frozen importlib._bootstrap>:1324, in _find_and_load_unlocked(name,␣
 ↪import_)

ModuleNotFoundError: No module named 'watermark'
```

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.

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.

1. Face Detection and Matching Overall Goal: The goal is to identify and match human faces in images or videos. Face detection identifies where the faces are out of the provided images or videos, while face matching is used to verify if two different faces belong to the same person. The applications of face detection and matching can be seen in security systems, biometric authentication, social tagging, and many more.

Appropriate Dataset It would include images or video frames with annotations indicating the location of faces, providing labeled identities to match. A typical dataset may feature thousands of images of people provided under variation in lighting, camera viewing angle, facial expression, and background.

Machine Learning Category: The problem falls into the category of supervised learning, where models are trained on labeled datasets such as face images with their associated labeled identities.

Performance Evaluation: You would validate your model based on the efficiency in face detection and matching. Success would be if it identifies and matches faces from new, unseen images and videos correctly. The performance could be measured in terms of accuracy, precision, and recall.

2. Sports Predictions Overall Goal: The aim, in essence, is to predict the outcomes of the sporting events, winners, scores, or even performance by players. This can then be used in betting, fantasy sports, or even optimizing team strategy.

Appropriate Dataset: A dataset that would best fit the requirements would be one that includes historical sports data such as statistics of teams, performance of players, game results, weather conditions, and maybe even expert predictions. It could also, if possible, include some sort of time-series data for each sport so that one could analyze the effect of the trends over seasons.

Machine Learning Category: In most cases, this falls under supervised learning if historical data is used to predict future outcomes. However, when one is designing a system that learns autonomously from experience to make better predictions based on feedback received from the real world, then this could even be a case of reinforcement learning, something like a betting model.

Performance Evaluation: You would evaluate the model by its performance at predicting the outcomes of previously unseen games. Success could be measured with a variety of metrics including accuracy in correct outcome prediction, mean squared error for predicted scores, and return on investment for betting.

3. Product Recommendations Overall Goal: The recommendation would go to the users based on their preferences, past behaviors, and possibly other users' behaviors. This could enrich user experience, enhance sales, and improve customer retention in e-commerce platforms.

Dataset: The dataset shall contain user interaction data, such as clicks, purchases, ratings, product data, and perhaps user demographics. In general, the size of the dataset is usually very large, with thousands of user-product interactions.

Category of Machine Learning: This could be a problem for both supervised and unsupervised learning: most collaborative filtering is unsupervised, since it relies on the similarities between users and items, while some are supervised, as there exist models that predict certain user preferences based on their past behavior. Some systems might even use reinforcement learning to adjust recommendations over time, based on user feedback.

Performance Evaluation: You would test your model on how well it makes recommendations that fit the preferences of a user. Success would be measured in terms of user engagement, for example, click-through rates, purchase rates, or more advanced metrics such as precision, recall, and mean reciprocal rank.

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]).

Spam classification depending only on the accuracy of classification can be indeed misleading, particularly for imbalanced data sets. It is important that a model could get high accuracy since it has to classify most emails as non-spam, with the majority probably being right while it misses lots of spam. There are pitfalls: false positives and false negatives wherein spam emails may slip through. These may be improved with better metrics including precision (minimum false positives), recall (minimum false negatives), the F1-score (effective balance between precision and recall), and ROC-AUC for measuring discrimination capability.

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).

1- Market Customer Segmentation : It ranges from segmenting customers based on purchasing behavior to demographics and interaction patterns. The task becomes a clustering task as no predefined labels are given, and hence, natural groups need to be identified. In contrast, unsupervised learning does not try to learn the nature of some intrinsic structure in the data but rather extracts information from the data, which is unlabeled in nature.

2- Document Clustering in Text Mining: This is a task that groups documents into groups of similar content or themes, for example, news articles, and research papers. This will be taken as a clustering task where we identify document similarities and form groups without any prior labeling being provided. No such predetermined categories have been there and an algorithm automatically identifies patterns and groups getting formed within the data. Hence, it's separated from supervised learning into another discipline.

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.

In the k-nearest neighbor (k-NN) algorithm, no actual computation happens during training because it is a lazy learning algorithm that simply stores the training data. At test time, the algorithm computes the distances between the test instance and all training instances to identify the k nearest neighbors, and then predicts the label based on these neighbors.

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.

In k-nearest neighbor (k-NN), adding more feature variables (dimensions) often leads to worse performance if the number of training examples is fixed. This is due to the curse of dimensionality, where the distance between points in higher-dimensional space becomes less meaningful because the data points become more spread out. As a result, the nearest neighbors may not be as "near," reducing the effectiveness of the algorithm. Additionally, with more features and the same number of training examples, the model may struggle with overfitting or capturing noise

1.6 E 6)

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

If your dataset contains noisy examples or outliers, it is generally better to increase k in the k-nearest neighbor (k-NN) algorithm. By increasing k, the algorithm considers more neighbors when making predictions, which helps average out the effect of noisy or outlier points. A larger k reduces the influence of any single noisy data point, leading to more robust and stable predictions. Conversely, a smaller k might allow the algorithm to be more sensitive to outliers, resulting in less accurate predictions

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.

[ ]:
```python
# 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
```

[ ]:
```python
def kronecker_delta(i, j):
# ENTER YOUR CODE HERE
    if(i==j):
            return 1
    else:
        return 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).

[ ]:
```
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

correct = 0
for true, pred in zip(y_true, y_pred):
    if true == pred:
        correct += 1

acc = (correct / len(y_true)) * 100

print('Accuracy: %.2f%%' % acc)
```

```
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]])
```

[ ]:
```
import numpy as np

# ENTER YOUR CODE HERE
mat = np.arange(9).reshape(3, 3)

print("'''python\narray(",mat)
```

```
'''python
array( [[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]])
```

[ ]:
```
# ENTER YOUR CODE HERE
np.random.seed(123)

arr = np.random.randn(2, 2)

print("'''python\narray(",arr)
```

```
'''python
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

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

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

1.12 E 12)

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

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

```
# ENTER YOUR CODE HERE
freq = np.bincount(A).argmax()
print("The most frequent integer is:",freq)
```

```
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

df_train = pd.read_csv("C:\\Users\\T L S\\Desktop\\Green␣
 ↪Vision\\FINAL\\train_data.txt")
df_train.head()
```

[ ]:
```
         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
```

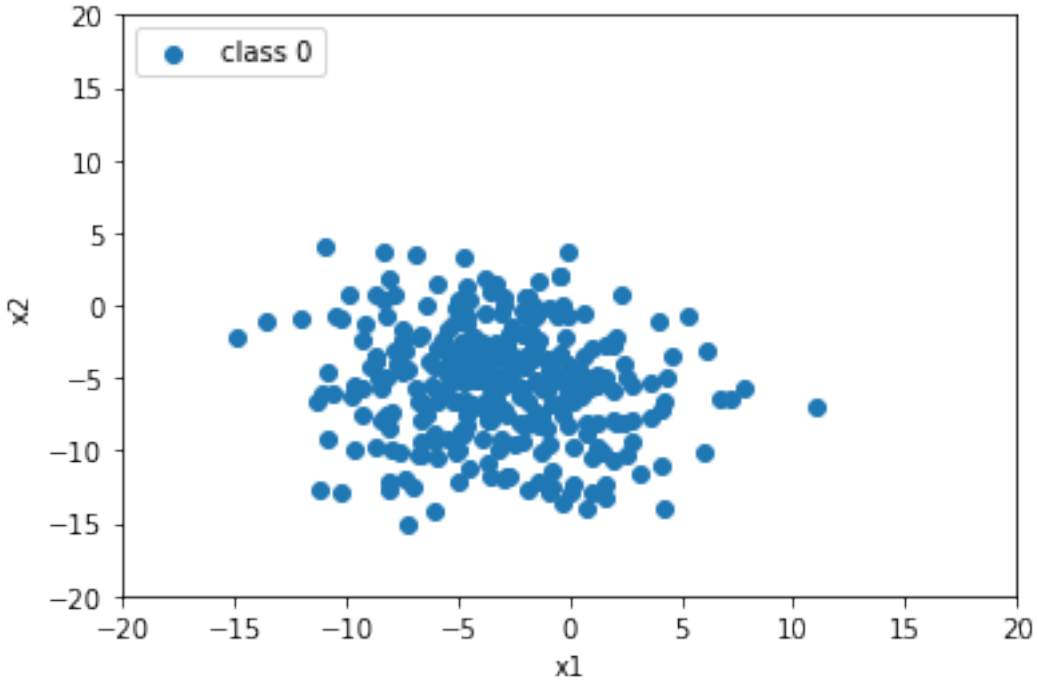
1.14 E 14)

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

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

[ ]:
```
%matplotlib inline
import matplotlib.pyplot as plt


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.show()
```



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',)

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()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[11], line 2
      1 import matplotlib.pyplot as plt
----> 2 plt.scatter(X_train[y_train == 0, 0],
      3             X_train[y_train == 0, 1],
      4             label='class 0',)
      6 plt.scatter(X_train[y_train == 1, 0],
      7             X_train[y_train == 1, 1],
      8             label='class 1')
     10 plt.xlabel('x1')

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

1.15 E 15)

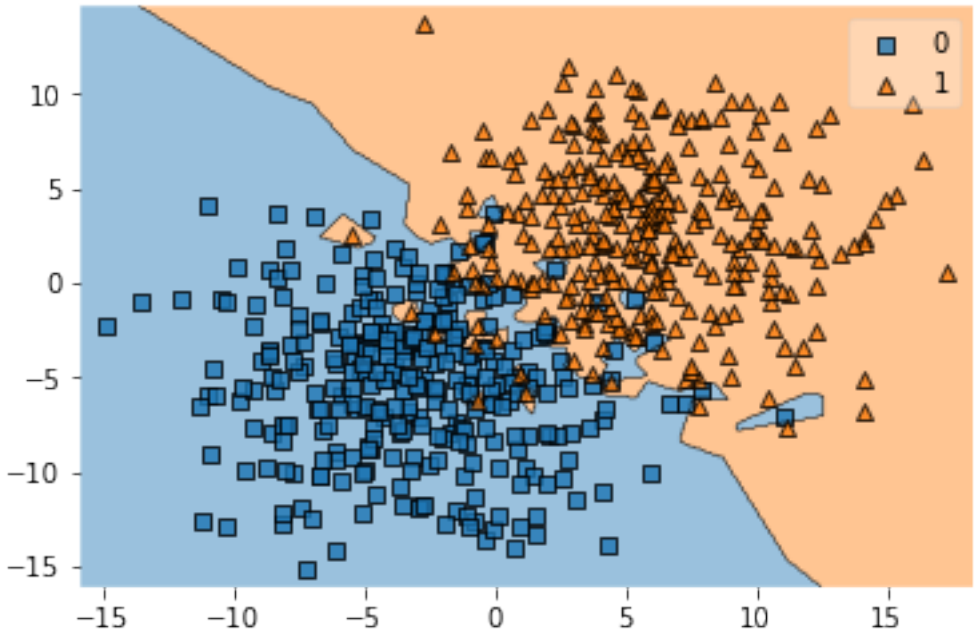
Consider the we trained a 1-nearest neighbor classifier using scikit-learn on the previous training dataset:

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

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

[ ]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=1, p=2,
           weights='uniform')
```

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

plot_decision_regions(X_train, y_train, knn)
```

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



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

[ ]:
```
from sklearn.metrics import accuracy_score

y_pred = knn.predict(X_train)

accuracy = accuracy_score(y_train, y_pred)

misclassification_error = 1 - accuracy

print("Misclassification Error:", misclassification_error)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[13], line 4
      1 from sklearn.metrics import accuracy_score
      3 # Step 1: Make predictions on the training set
----> 4 y_pred = knn.predict(X_train)
      6 # Step 2: Calculate accuracy
      7 accuracy = accuracy_score(y_train, y_pred)

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

1.16 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

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from mlxtend.plotting import plot_decision_regions
from sklearn.metrics import accuracy_score


k_values = [1, 3, 5, 7, 9]

fig, axes = plt.subplots(1, 5, figsize=(25, 5))


for idx, k in enumerate(k_values):

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


    plot_decision_regions(X_train, y_train, clf=knn, ax=axes[idx])
    axes[idx].set_title(f'k = {k}')
    axes[idx].set_xlim([-20, 20])
    axes[idx].set_ylim([-20, 20])


    y_pred = knn.predict(X_train)
    accuracy = accuracy_score(y_train, y_pred)
    misclassification_error = 1 - accuracy


    print(f"Misclassification Error for k={k}: {misclassification_error:.4f}")


plt.tight_layout()
plt.show()
```

1.17 E 17)

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

[17]:
```
df_test = pd.read_csv("C:\\Users\\T L S\\Desktop\\Green Vision\\FINAL\\test_data.txt")

df_test.head()
```

[17]:

	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):

[ ]:
```
X_test = df_test[['x1', 'x2']].values
y_test = df_test['y'].values
```

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.

[ ]:
```
from sklearn.model_selection import train_test_split


X_train_sub, X_val, y_train_sub, y_val = train_test_split(X_test, y_test, test_size=0.3,random_state=123, stratify=y_train)
```

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.

[ ]:
```
for k in range(1, 15):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_sub, y_train_sub)
    # ENTER YOUR CODE HERE
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score


for k in range(1, 15):
    knn = KNeighborsClassifier(n_neighbors=k)
```

```
    knn.fit(X_train_sub, y_train_sub)


    y_train_pred = knn.predict(X_train_sub)
    y_val_pred = knn.predict(X_val)
    y_test_pred = knn.predict(X_test)


    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(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)
```

1.20 E 20)

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

[ ]:
```
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
```

[ ]:
```
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 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:

[ ]:
```
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:

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

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

[ ]:
```
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)
```

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.

[ ]:
```
import numpy as np

def euclidean_dist_vectorized(a, b):
    return np.sqrt(np.sum((a - b) ** 2))

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

%timeit euclidean_dist_vectorized(a, b)
```

Using np.linalg.norm

[ ]:
```
import numpy as np

def euclidean_dist_norm(a, b):
    return np.linalg.norm(a - b)
```

```
%timeit euclidean_dist_norm(a, b)
```

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`:

[ ]:
```
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:

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

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

[ ]:
```
import numpy as np

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

%timeit sorted(c)[:3]
```

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:

[ ]:
```
from heapq import nsmallest

nsmallest(3, c)
```

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

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

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

%timeit nsmallest(3, c)
```

Summarize your findings in 1-3 sentences.

Using the `sorted` function to find the smallest `k` values in an array is less efficient because it requires sorting the entire array, which can be slow for large datasets. In contrast, the `heapq.nsmallest` function, which uses a heap data structure, provides a faster and more efficient way to find the

smallest `k` values without fully sorting the array. Benchmarking shows that `nsmallest` significantly outperforms the full sort approach in terms of execution time.