Exercise 3: Learning from Data

In this exercise, you will train different types of regression and classification models on two datasets. You will be graded based on a combination of your code producing the expected results, your written responses to the questions and a passing leaderboard score in the final exercise.

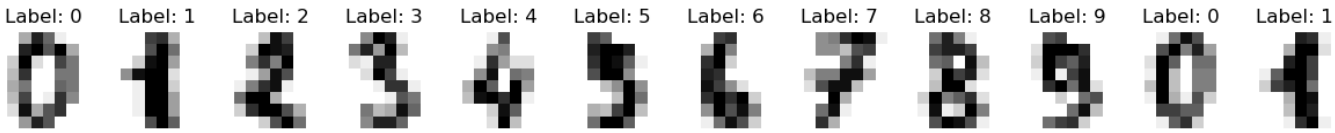
3.1 Linear / Logistic Regression for digit recognition

In this part you will experiment with two different types of prediction models, and qualitatively + quantitatively compare them. You will be working with the classic MNIST dataset, which we can load from `sklearn.datasets`.

In [222…
```
# Importing the dataset
from sklearn import datasets
digits = datasets.load_digits();
```

Here we plot the first few digits and their labels. Apparently they are in order, but our upcoming models will not rely on this.

In [223…
```
import matplotlib.pyplot as plt

_, axes = plt.subplots(nrows=1, ncols=12, figsize=(15, 3))
for ax, image, label in zip(axes, digits.images, digits.target):
    ax.set_axis_off()
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation="nearest")
    ax.set_title("Label: %i" % label)
```



A: Warmup Exercise

Use the `train_test_split` function from `sklearn.model_selection` twice to split the data into train, validation, and test sets, with a distribution of 80%, 10%, and 10% respectively. Use `random_state=0` for reproducibility. Check out the documentation to figure out how to set the parameters to get the desired split.

In [224…
```
# INSERT YOUR CODE HERE. ASSIGN TO THE FOLLOWING VARIABLES, X_train, X_val, X_test, y
from sklearn.model_selection import train_test_split

data = digits.images.reshape((len(digits.images), -1))

X_train, X_test, y_train, y_test = train_test_split(
    data,
    digits.target,
    test_size=1/10,
    shuffle=False,
    random_state=0
)

X_train, X_val, y_train, y_val = train_test_split(
    X_train,
    y_train,
    test_size=1/9,
```

```
    shuffle=False,
    random_state=0
)

print(len(X_train), len(y_train))
print(len(X_val), len(y_val))
print(len(X_test), len(y_test))
```

```
1437 1437
180 180
180 180
```

Now let's check out the returned training data, which is of the type `numpy.ndarray`, which you might have seen already. We want to make sure the classes are balanced, e.g. we don't have a lot more 4's than 5's.

- How many images do we have for the train, validation, and test sets?
- What is the distribution of the training labels? Comment on whether the classes are balanced.

In [225…

```
# CALCULATE AND WRITE YOUR ANSWERS HERE
import numpy as np

print(f'{len(y_train)} training images')
print(f'{len(y_val)} validation images')
print(f'{len(y_test)} test images')

for i in range(10):
    amount = np.count_nonzero(y_train == i)
    print(f'{amount} of label {i} in training set')
```

```
1437 training images
180 validation images
180 test images
143 of label 0 in training set
146 of label 1 in training set
142 of label 2 in training set
146 of label 3 in training set
144 of label 4 in training set
145 of label 5 in training set
144 of label 6 in training set
143 of label 7 in training set
141 of label 8 in training set
143 of label 9 in training set
```

The classes are fairly balanced as to not cause problems or biases.

B: Programming Exercise

Let's build a model that learns to predict the digit given the image. We'll first try our hand at linear regression. We will use the `LinearRegression` class from `sklearn.linear_model`. Again, check out the documentation on how to use it. We will use the `fit` method to train the model, and the `predict` method to make predictions on the test set. Afterwards, evaluate the model using functions from `sklearn.metrics` module.

In [226…

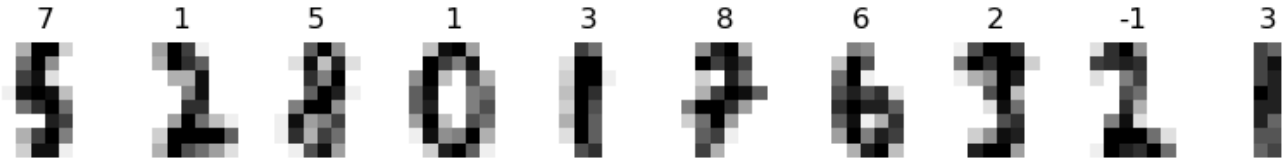
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, accuracy_score

# DEFINE A LinearRegression MODEL USING SKLEARN. ASSIGN THE MODEL TO THE VARIABLE mod
linear_model = LinearRegression()

# FIT THE MODEL AND MAKE PREDICTIONS ON THE TEST SET. ASSIGN THE PREDICTIONS TO THE V
linear_model.fit(X_train, y_train)
```

```
# ASSIGN YOUR MODEL'S TEST SET MSE TO THE VARIABLE mse AND ITS ACCURACY TO THE VARIAB
y_pred = linear_model.predict(X_test)
mse = format(mean_squared_error(y_test, y_pred), '.2f')
acc = format(accuracy_score(y_test, np.round(y_pred)), '.2f')
```

Should we not be using the X_val and y_val for something?

In [227…
```
# DO NOT INSERT OR CHANGE ANYTHING BELOW
print("LinearRegression MSE: ", mse)
print("LinearRegression accuracy: ", acc)

_, axes = plt.subplots(nrows=1, ncols=10, figsize=(10, 3))
for ax, image, prediction in zip(axes, X_test, np.round(y_pred)):
    ax.set_axis_off()
    image = image.reshape(8, 8)
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation="nearest")
    ax.set_title("%i" % np.round(prediction))
```

```
LinearRegression MSE:  4.81
LinearRegression accuracy:  0.15
```



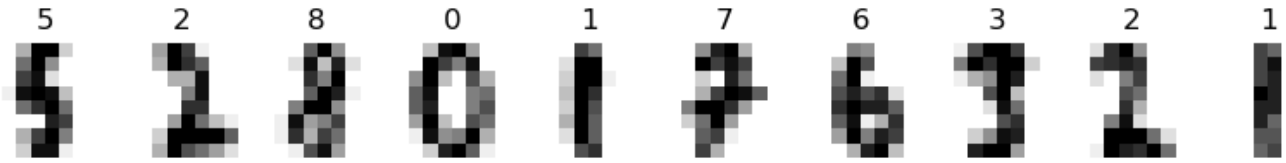
C: Programming Exercise

Now, let's model the task differently. You will use the `LogisticRegression`, again included in the module `sklearn.linear_model`, and again, check out the documentation to see how to use it. Fit the model and make predictions on the test set.

You are asked to produce output with the same structure as in the above programming exercise (two printouts and one image), but for a different model this time.

In [228…
```
# YOUR CODE HERE
from sklearn.linear_model import LogisticRegression

# Training
logistic_model = LogisticRegression(max_iter=1000)
logistic_model.fit(X_train, y_train)

# Evaluation
y_pred = logistic_model.predict(X_test)
mse = format(mean_squared_error(y_test, y_pred), '.2f')
acc = format(accuracy_score(y_test, y_pred), '.2f')

print("LogisticRegression MSE: ", mse)
print("LogisticRegression accuracy: ", acc)

_, axes = plt.subplots(nrows=1, ncols=10, figsize=(10, 3))
for ax, image, prediction in zip(axes, X_test, np.round(y_pred)):
    ax.set_axis_off()
    image = image.reshape(8, 8)
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation="nearest")
    ax.set_title("%i" % np.round(prediction))
```

```
LogisticRegression MSE:  1.67
LogisticRegression accuracy:  0.93
```



Written Questions

1a) Describe the trends you observe in the results. Is there anything interesting or unexpected?

This model is a lot better. I am still confused about why we use MSE for such a model though, since I calculate it by taking the amount of numbers it off by. Accuracy makes a lot of sence though.

1b) How do the results compare to the linear regression model?

The logistic regression model is a lot more accurate.

1c) What do you think is the reason for this?

Because linear is used for continuos values for regression models, while logistic is used for category values for categorial models like in this problem.

Exercise 3.2: Determine habitability

Prediction task: Build a logistic model for exoplanet habitability

Your team is tasked with applying data science and machine learning techniques to analyze exoplanet data from the famous NASA Exoplanet Archive (NEA). Among 372 features, your goal is to build a model that predicts whether an exoplanet is potentially habitable (1) or non-habitable (0).

Evaluation metric: F1-score

Due to the highly imbalanced nature of the data—habitable planets are much rarer than their non-habitable counterparts—the evaluation metric is the F1-score, which balance precision and recall. This helps us know whether your model is truly effective at identifying the minority (habitable) class rather than simply achieving high overall accuracy by predicting the majority class.

Feature engineering

The most minimal version of a working solution should give you around 50-60% F1.

You can improve this performance substantially—achieve closer to **85-90%** F1—by feature engineering and data preprocessing:

- **Feature selection:** Only choosing relevant features (Hint: Look out for anything temperature-related)
- **Feature creation:** Earth Similarity Index (see below)
- **Feature transformation:** Scaling and standardization
- **Imputation** of NaN/unknown values
- **Oversampling** to increase the minority class

Consider conducting exploratory data analysis (EDA) to gain useful insight about the dataset and leveraging a Pipeline to streamline your steps. For more information see the description on the Kaggle page.

Good luck, and we look forward to seeing your solutions!

D: Programming Exercise

1. Create an account on Kaggle to join the Kaggle competition: GDS Exercise 3 – 2025. You can either use your real name, or tell us your Kaggle username, so we can grade you.

Kaggle name: 'Victor V. Jørgensen' eller 'victorvjrgensen'

2. Download the training and test data. Check out the Dataset Description for more information about the data and your task.
3. Load the training data file `train.npz` and split it into `train`, `val`, and `test` splits.
4. Use a linear or logistic regression model to fit the `train` split. Use the `val` split for model selection and hyperparameter optimization, while reserving the `test` split for evaluation exclusively. (You should measure performance using f1-score).

We load our data:

In [229…
```
import pandas as pd

df = pd.read_csv('train.csv')
df
```

Out[229…

	rowid	hostname	pl_letter	hd_name	hip_name	tic_id	gaia_id	de
0	2169.0	HD 13908	b	HD 13908	HIP 10743	TIC 12906712	Gaia DR2 515574853541230720	
1	6944.0	Kepler-1025	b	NaN	NaN	TIC 164727439	Gaia DR2 2106595941599427072	
2	6385.0	KOI-55	b	NaN	NaN	TIC 184427882	Gaia DR2 2076819620539690880	
3	30104.0	Kepler-673	b	NaN	NaN	TIC 417657148	Gaia DR2 2130040450081573376	
4	3198.0	HD 37124	c	HD 37124	HIP 26381	TIC 19631691	Gaia DR2 3402798414192387328	
…	…	…	…	…	…	…	…	
4474	18711.0	Kepler-196	b	NaN	NaN	TIC 164828012	Gaia DR2 2104094071610116608	
4475	5333.0	K2-383	b	NaN	NaN	TIC 31244979	Gaia DR2 4098446119569242752	
4476	14321.0	Kepler-1565	b	NaN	NaN	TIC 170241112	Gaia DR2 2073793249146651136	
4477	3821.0	HIP 21152	b	HD 28736	HIP 21152	TIC 452767166	Gaia DR2 3285426613077584384	
4478	22653.0	Kepler-296	f	NaN	NaN	TIC 243271945	Gaia DR2 2132069633148965888	

4479 rows × 373 columns

I use the following to find features that I like that also has a lot of data points.

In [230…
```
pd.DataFrame(df.count()).sort_values(by=0, ascending=False)
```

Out[230…

	0
P_HABITABLE	4479
P_INCLINATION_LIMIT	4479
S_MASS_LIMIT	4479
S_TEMPERATURE_LIMIT	4479
S_DEC_STR	4479
...	...
pl_occdep	2
pl_occdeperr2	1
pl_occdeperr1	1
sy_kepmagerr2	0
sy_kepmagerr1	0

373 rows × 1 columns

I've selected some features I like and process the columns of theese features into arrays of non-zero values by flattening the values on NaN values.

X_names has the data without the labels.

y_names contains the labels for X_names.

In [237…

```
from sklearn.impute import SimpleImputer

chosen_columns = [
    'P_TEMP_SURF',
    'S_TEMPERATURE',
    'P_TEMP_SURF_MIN',
    'P_TEMP_SURF_MAX',
    'P_TEMP_EQUIL',
    'P_TEMP_EQUIL_MIN',
    'P_TEMP_EQUIL_MAX',
    'S_TEMPERATURE_ERROR_MIN',
    'S_TEMPERATURE_ERROR_MAX',
    'S_AGE',
    'S_AGE_ERROR_MIN',
    'S_AGE_ERROR_MAX',
    'P_MASS',
    'P_FLUX',
    ]

def data_cleaner(data):
    data_cleaned = data.copy()
    for column in chosen_columns:
        imputer = SimpleImputer(strategy='mean')
        data_cleaned[column] = pd.Series(imputer.fit_transform(data_cleaned[[column]]
    return data_cleaned

def X_names(data):
    data_cleaned = np.stack([data_cleaner(data)[column] for column in chosen_columns]
    return data_cleaned

def y_names(data):
    data_cleaned = np.array(data_cleaner(data)['P_HABITABLE'])
    return data_cleaned
```

```
pd.DataFrame(data_cleaner(df))
```

Out[237…

	rowid	hostname	pl_letter	hd_name	hip_name	tic_id	gaia_id	de
0	2169.0	HD 13908	b	HD 13908	HIP 10743	TIC 12906712	Gaia DR2 515574853541230720	
1	6944.0	Kepler-1025	b	NaN	NaN	TIC 164727439	Gaia DR2 2106595941599427072	
2	6385.0	KOI-55	b	NaN	NaN	TIC 184427882	Gaia DR2 2076819620539690880	
3	30104.0	Kepler-673	b	NaN	NaN	TIC 417657148	Gaia DR2 2130040450081573376	
4	3198.0	HD 37124	c	HD 37124	HIP 26381	TIC 19631691	Gaia DR2 3402798414192387328	
…	…	…	…	…	…	…	…	
4474	18711.0	Kepler-196	b	NaN	NaN	TIC 164828012	Gaia DR2 2104094071610116608	
4475	5333.0	K2-383	b	NaN	NaN	TIC 31244979	Gaia DR2 4098446119569242752	
4476	14321.0	Kepler-1565	b	NaN	NaN	TIC 170241112	Gaia DR2 2073793249146651136	
4477	3821.0	HIP 21152	b	HD 28736	HIP 21152	TIC 452767166	Gaia DR2 3285426613077584384	
4478	22653.0	Kepler-296	f	NaN	NaN	TIC 243271945	Gaia DR2 2132069633148965888	

4479 rows × 373 columns

Vi split out training data into 90% training and 10% test sets.

We then look at the distribution of habitable planets.

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X_names(df),
    y_names(df),
    test_size=0.1,
    shuffle=False,
    random_state=0,
)

print(len(X_train), len(y_train))
print(len(X_test), len(y_test))

print(f'Habitable amount in training set: {np.count_nonzero(y_train == 1)}')
print(f'Unhabitable amount in training set: {np.count_nonzero(y_train == 0)}')

print(f'Habitable amount in test set: {np.count_nonzero(y_test == 1)}')
print(f'Unhabitable amount in test set: {np.count_nonzero(y_test == 0)}')
```

```
4031 4031
448 448
Habitable amount in training set: 46
Unhabitable amount in training set: 3985
Habitable amount in test set: 10
Unhabitable amount in test set: 438
```

We train out logistic regression model and evaluate it.

In [234…
```
from sklearn.metrics import f1_score

# Training
model = LogisticRegression(max_iter=10000)
model.fit(X_train, y_train)

# Evaluation
y_preds = model.predict(X_test)
f1 = format(f1_score(y_test, y_preds), '.2f')
print("Model f1: ", f1)
```

```
Model f1:  0.80
```

We got a f1 score of 0.8 on our test set.

E: Calculate the Earth Similarity Index (Optional)

Evidently, astronomers have observed Earth to be a very habitable planet (1) with their telescopes. As such, we make the remark that planets with properties similar to Earth tend to make excellent candidates for planet habitability.

Earth Similarity Index (ESI) is a metric defined as:

$$\mathrm{ESI} = \prod_{i=1}^{n} \left(1 - \left|\frac{x_i - x_{i,\oplus}}{x_i + x_{i,\oplus}}\right|\right)^{\frac{w_i}{n}}$$

where
x_i and $x_{i,\oplus}$ are features of the extraterrestrial body and of Earth respectively,
w_i is the weighted exponent of each feature, and
n is the total number of features.

The weight assigned to each feature, w_i, are free parameters that can be chosen to emphasize certain characteristics over others, e.g. surface temperature is often given a much higher weight because maintaining liquid water is crucial.

1. For each selected feature, determine w_i. You may default these to 1 (equal weight) or draw on domain knowledge.
2. Compute the `ESI` for each exoplanet in `X_train` and `X_test`. Check whether your results approximately match those in `exampleESI.csv`
3. When confident in you calculations, add `ESI` as a new column in both sets.
4. Retrain your logistic model with your new `ESI` feature included.

In [235…
```
# YOUR CODE HERE
# >
```

F: Unlabelled data evaluation

After downloading the unlabeled test.npz, you will make predictions in `y_pred` for all of the unlabeled exoplanets in `X_test`. You can submit your best predictions to Kaggle three times/day.

This programming exercise is considered **passed** if you achieve $\text{F1-Score} \geq 0.80$ on the leaderboard using only a logistic model.

In [236…

```
# LOAD THE TEST SET AND MAKE PREDICTIONS. ASSIGN THE PREDICTIONS TO THE VARIABLE y_pr
test_data = pd.read_csv('test.csv')

y_pred = model.predict(X_names(test_data))

sample = pd.read_csv('sample_submission.csv') # Load `sample_submission.csv` to valid
sample['P_HABITABLE'] = y_pred  # assuming y_pred contains your predictions
sample.to_csv('submission.csv', index=False)

# NOW YOU CAN UPLOAD THE submission.csv FILE TO KAGGLE AND SEE YOUR ACCURACY
```

Written Questions

2a) Explain the different techniques and their purpose in your code for Programming Exercise D.

I have made some comments above my code cells that explains my thoughts.

To sum up my code, I sorted the feature columns by data point amount, which made it easy to find features with a lot of data. I decided to make use of NaN values by flattening my values over theese NaN values, this made it possible to use otherwise unusable data. Everything else is nothing new from the warmup exercise A.

2b) Which techniques were not useful in improving performance on this task?

I tried to use label encoders to get string features, but I found a lot of data that was represented in both string and float values. For example, I found a feature that contained the string description of the planets temperature like: 'Hot', 'Warm', 'Cold', but then I found even more accurate features that had the actual temperature estimates. So I avoided label encoders for this assignment.

2c) Optional: Consider the exoplanet `Kepler-438 b` with `ESI=0.832838` and `P_HABITABLE=0`. Despite its high ESI, this planet gets violently nuked by its own sun every three Earth months. Discuss how a loss of granularity when designing a composite index, like the ESI, can result in masking outlier behavior.