

# 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

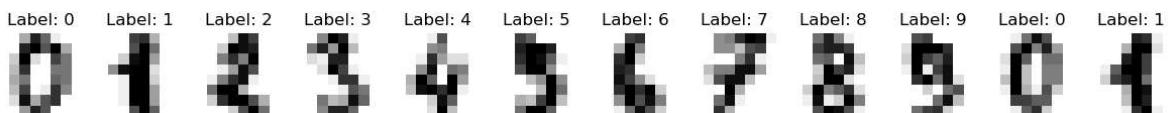
In this part you will experiment with two different types of prediction models, and qualitatively + quantitatively compare them. You will be working with the classic MNIST dataset, which we can load from `sklearn.datasets`.

```
In [ ]: # 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 [ ]: 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 [ ]: # INSERT YOUR CODE HERE. ASSIGN TO THE FOLLOWING VARIABLES, X_train, X_val, X_test,
# >
from sklearn.model_selection import train_test_split
(X_train, X_temp, y_train, y_temp) = train_test_split(digits.data, digits.target, 
(X_val, X_test, y_val, y_test) = train_test_split(X_temp, y_temp, test_size=0.5, r
```

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 [ ]: # CALCULATE AND WRITE YOUR ANSWERS HERE
# >
print('# of images:')
print('X_train : ' + str(X_train.shape[0]))
print('X_test : ' + str(X_test.shape[0]))
print('X_val : ' + str(X_val.shape[0]))

print(" ")
print("distribution")
print('X_train : ' + str(round(((X_train.shape)[0] / digits.data.shape[0] * 100))))
print('X_test : ' + str(round(((X_test.shape)[0] / digits.data.shape[0] * 100))) +
print('X_val : ' + str(round(((X_val.shape)[0] / digits.data.shape[0] * 100))) + '%')

# of images:
X_train : 1437
X_test : 180
X_val : 180

distribution
X_train : 80%
X_test : 10%
X_val : 10%
```

## 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 [ ]: # DEFINE A LinearRegression MODEL USING SKLEARN. ASSIGN THE MODEL TO THE VARIABLE r
# >
import numpy as np
import sklearn.linear_model
model = sklearn.linear_model.LinearRegression()

# FIT THE MODEL AND MAKE PREDICTIONS ON THE TEST SET. ASSIGN THE PREDICTIONS TO THE VARIABLE y_pred
# >
model.fit(X_train, y_train)
y_pred = ([round(x) for x in model.predict(X_test)])
```

```
# ASSIGN YOUR MODEL'S TEST SET MSE TO THE VARIABLE mse AND ITS ACCURACY TO THE VARIABLE acc
# >
from sklearn.metrics import mean_squared_error
from sklearn.metrics import accuracy_score

mse = mean_squared_error(y_test, y_pred)
acc = accuracy_score(y_test, y_pred)

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

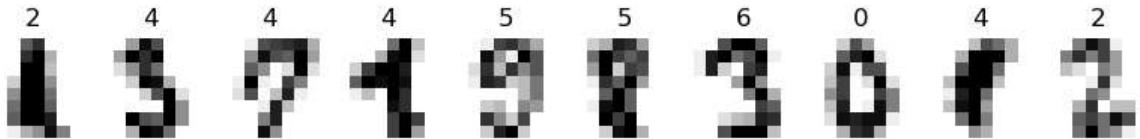
`_`, axes = plt.subplots(nrows=1, ncols=10, figsize=(10, 3))

```

for ax, image, prediction in zip(axes, X_test, 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" % prediction)

```

LinearRegression MSE: 3.088888888888888  
 LinearRegression accuracy: 0.22777777777777777



## 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 [ ]:

```

# YOUR CODE HERE
# >
modelR = sklearn.linear_model.LogisticRegression(max_iter=2170)
modelR.fit(X_train, y_train)
y_predR = ([x for x in modelR.predict(X_test)])


mseR = mean_squared_error(y_test, y_predR)
accR = accuracy_score(y_test, y_predR)
print(mseR)
print(accR)

```

1.138888888888888  
 0.9611111111111111

## Written Questions

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

- From these result we see that: the lower the MSE the higher the accuracy, and vice versa

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

- The results of logistic regression were remarkably better than the linear regression model.

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

- Since we're trying to classify different numbers logistic regression worked best, if we were to predict a numeric value

based on trends Linear regression would work best

## Exercise 3.2: Counting stars

Your team is tasked with applying data science and machine learning techniques to some recently collected telescope image data. The telescope takes low resolution images, in which stars either appear as squares or circles. Your task is to train a classifier to determine whether there are more circular stars than square stars in each image. The budget is tight this year so there are no resources available to train expensive deep neural networks; you will need to train a linear or logistic regression classifier instead.

You are encouraged to be creative, and to try out different things empirically, too. Preprocessing could be crucial to the success of your model. Good luck!

Below is a sample of the data you'll be working with. Above each image is the label, which shows the number of circular and square stars, respectively. We do not count the small noisy background stars.

## D: Programming Exercise

1. Create an account on Kaggle to collect the data  
<https://www.kaggle.com/competitions/ds2023-counting-stars/>. Use your real name, or tell us your Kaggle username, so we can grade you.
2. Download the training data from <https://www.kaggle.com/competitions/ds2023-counting-stars/data>. Check out the Dataset Description for more information about the data and your task.
3. Load the training data file `stars_train_X_y.npz` and split it into `train`, `val`, and `test` splits.
4. Use a linear or logistic regression model to fit the `train` data. Use the `val` and `test` splits for model selection and hyperparameter optimization. (You should measure performance using accuracy).

```
In [ ]: import sklearn.linear_model
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import accuracy_score

##### -- Load data --
data = np.load('stars_train_X_y.npz')
X = data['X_train'] # np.ndarray of size (8000, 4096), flattened image (64x64)
y = data['y_train'] # np.ndarray of size (8000, 2), number of circular stars and

##### -- Preprocessing --
''' Convert to binary list
1 if there is more circular stars, 0 otherwise '''
def convertBinary(input):
    lst = []
    for i in range(0, len(input)):
        if (input[i][0]) > (input[i][1]):
            lst.append(1)
```

```

        else: lst.append(0)
    return lst
y = convertBinary(y)

'''Removes noise'''
def lstPop(inp):
    lst = inp
    for i in range(len(lst)):
        if i == 0 or i == len(lst)-1: # check if first or last element
            continue
        if lst[i-1] == 0 and lst[i+1] == 0 and lst[i] != 0:
            lst[i] = 0
    return lst

''' Reverse dimentionality reduction'''
def divide(input):
    lst = []
    for i in range (0, len(input)):
        arr1 = []
        arr2 = []
        for j in range (0, len(input[i])):
            if input[i][j] == 0 or input[i][j] == 2:
                arr2.append(0)
            else:
                arr2.append(1)
            if input[i][j] == 0 or input[i][j] == 1:
                arr1.append(0)
            else:
                arr1.append(2)
        arr1 = lstPop(arr1)
        arr2 = lstPop(arr2)
        lst.append((arr2)+(arr1))
    return lst
X = (divide(X))

##### -- Split data -- #####
(X_train, X_temp, y_train, y_temp) = train_test_split(X, y, test_size=0.2, random_
(X_test, X_val, y_test, y_val) = train_test_split(X_temp, y_temp, test_size=0.5, r

```

In [ ]: `from sklearn.model_selection import GridSearchCV`

```

def gridSearch():
    hyperparameters = {
        'C': [0.001, 0.01, 0.1, 1, 10, 100],
        'penalty': ['l1', 'l2', 'elasticnet'],
        'solver': ['liblinear', 'saga'],
        'class_weight': ['balanced', None],
    }
    model = sklearn.linear_model.LogisticRegression()
    grid_search = GridSearchCV(model, hyperparameters, cv=5)
    grid_search.fit(X_train, y_train)

    print("Best hyperparameters: ", grid_search.best_params_)
    print("Test score: ", grid_search.score(X_val, y_val))
gridSearch()

```

In [ ]: `##### -- Prediction -- #####`

```

def predict(trainX_, trainY_, testX_, testY_ ):
    model = sklearn.linear_model.LogisticRegression(C = 0.01, penalty = 'l2', solver='liblinear')
    model.fit(trainX_, trainY_)
    pred = model.predict(testX_)
    MSE = mean_squared_error(testY_, pred)

```

```

ACC = accuracy_score(testY_, pred)
print("MSE : " + str(MSE))
print("ACC : " + str(ACC) )
return pred

prediction = predict(X_train, y_train, X_test, y_test)

def predictNonLabeled(trainX_, trainY_, testX_):
    model = sklearn.linear_model.LogisticRegression(C = 0.01, penalty = 'l2', solver='liblinear')
    model.fit(trainX_, trainY_)
    pred = model.predict(testX_)
    return pred

```

MSE : 0.045  
ACC : 0.955

## E: Unlabelled data evaluation

You should download the unlabeled test data from <https://www.kaggle.com/competitions/ds2023-counting-stars/data> and make predictions for all of the unlabeled images in the test set. You can submit your best predictions to Kaggle three times/day.

The course responsible has submitted a naive baseline and we're sure you can beat that score! This programming exercise is considered passed if you achieve Score  $\geq 0.90$  on the leaderboard.

```
In [ ]: # LOAD THE TEST SET AND MAKE PREDICTIONS. ASSIGN THE PREDICTIONS TO THE VARIABLE y_
# >

data = np.load('stars_test_X.npz')
testSet = data['X_test']
testSet = divide(testSet)
y_pred = predictNonLabeled(X_train, y_train, testSet)

output_file = "prediction.csv"
# DO NOT INSERT OR CHANGE ANYTHING BELOW
# Use this function to save your predictions on stars_test_X.npz to a "prediction.csv" file.
# The produced csv file can be directly uploaded to Kaggle for evaluation.
def y_pred_to_kaggle_format(y_pred : np.ndarray) -> None:
    np.c_[np.arange(1, len(y_pred)+1), y_pred]
    np.savetxt(output_file, np.c_[np.arange(1, len(y_pred)+1), y_pred], fmt='%i', delimiter=',')

y_pred_to_kaggle_format(y_pred)

# NOW YOU CAN UPLOAD THE prediction.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.**

### Preprocessing of X.

For every element in X, I did the following:

- Split the dataset into 2 64X64 arrays where in the 1st array, all the 2's were converted to 0, and in the second array, all the 1's were converted into 0's.
- I removed "lonely pixels" for both arrays to reduce noise. This resulted in slightly better accuracy.
- Then I concatenated the two arrays. The process of splitting and concatenation made a significant impact on the models' accuracy.  
</ul>

### **Preprocessing of y:**

Since I'm dealing with binary classification, I had to convert y into a binary list. Each element is equivalent to a 1 if there are more circular stars and a 0 if there are more square stars.

### **Grid search**

I used Gridsearch for hyperparameter turning.

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

Before using grid search, I tuned the hyper parameters by try and error. A lot of the parameters either didn't improve or decreased the accuracy.

### **Linear regression**

The accuracy decreased significantly when I used linear regression, but I should have suspected this because I was using it for binary classification. If I instead had used linear regression to predict the number of circular and square stars and compared the result, linear regression properly would be sufficient.