Name	Matr.Nr.	Due Date
Mohamed Abdelaziz	12137202	13.12.2021

Hands-on AI I

Unit 4 – Logistic regression as a door opener to Deep Learning

Authors: Brandstetter, Schäfl, Winter, Parada-Cabaleiro, Schörgenhumer
Date: 29-11-2021

In [2]:
```
# Required packages and the u4_utils file
import u4_utils as u4
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import torch

u4.check_module_versions()
# Set plotting style of seaborn related plots.
sns.set()
```

```
Installed Python version: 3.9 (✓)
Installed numpy version: 1.21.1 (✓)
Installed pandas version: 1.3.1 (✓)
Installed scikit-learn version: 1.0 (✓)
Installed matplotlib version: 3.4.3 (✓)
Installed seaborn version: 0.11.2 (✓)
Installed torch version: 1.10.0+cpu (✓)
```

Note: When specifying a seed for the sources of randomness, use the `u4.set_seed(seed=XYZ)` function.

Exercise 1

Given the dataset defined below, use the functions from `u4_utils.py` (according to the instructions given in the lecture notebook) to perform the following tasks:

Parameter	Value (used in this notebook)	Description
`num_pairs`	75	amount of (x, y) pairs to generate
`variance`	0.2	variance within y w.r.t. defining function

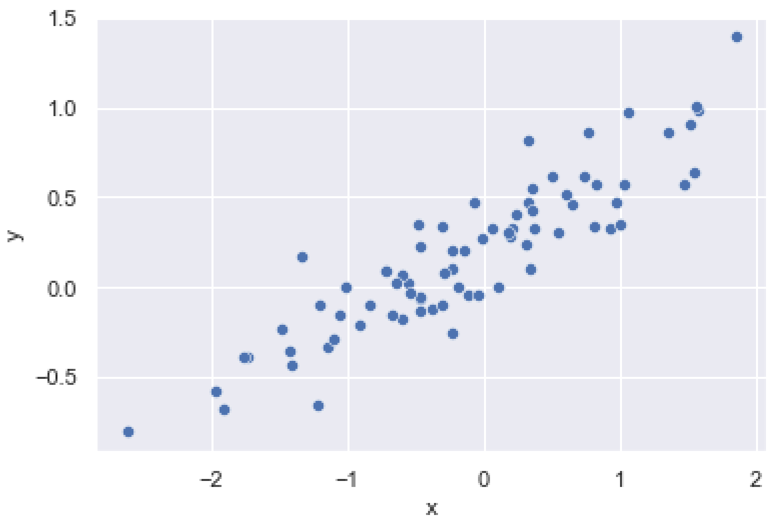
- In order to find the *linear* model which best describes the given dataset, look for the optimal parameters k and d manually. Then, plot both the data pairs and the linear model (defined by your coefficients).

Note: For reproducibility, set a fixed seed (seed=42).

- Perform the previous task but this time by setting the parameters 'automatically', in a way that the **Mean Squared Error** between the linear model and the data pairs is minimized. Again, plot both the data pairs and the linear model.

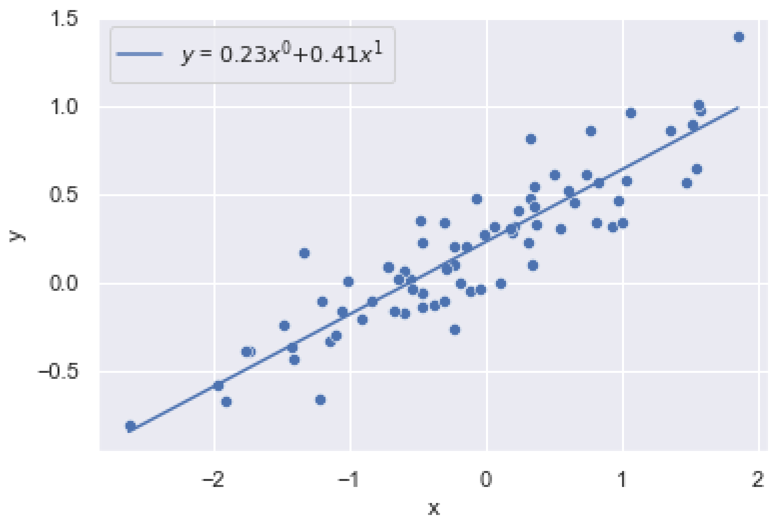
Note: After performing the second task, you might revise the answer you have given to the first one.

In [3]:
```
u4.set_seed(seed=42)

dataset = u4.get_dataset(num_pairs=75, variance=0.2)

sns.scatterplot(data=dataset, x="x", y="y");
```



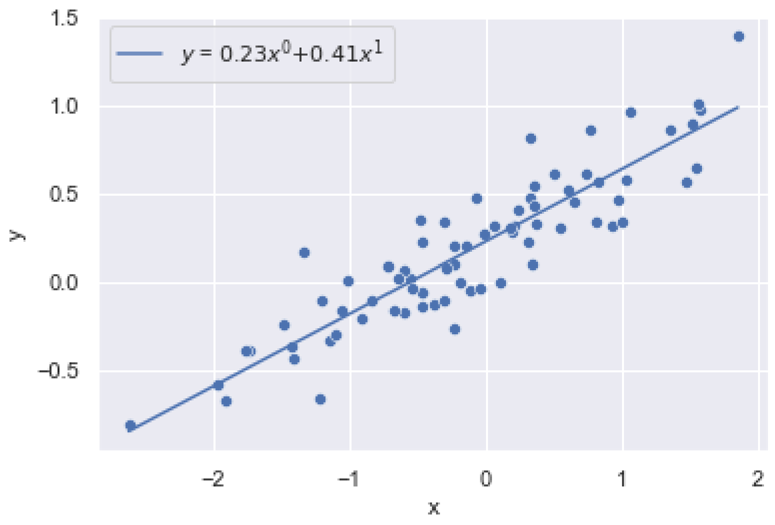
1.1. Manually define the coefficients of an underlying linear model. Plot data pairs as well as the defined linear model.

In [4]:
```
coefficients = (0.23,0.41)
u4.plot_model(dataset=dataset, coefficients=coefficients)
```



1.2. Determine the coefficients 'automatically'. Plot data pairs as well as the defined linear model.

In [5]:
```
coefficients = u4.minimize_mse(dataset=dataset, degree=1)
u4.plot_model(dataset=dataset, coefficients=coefficients)
```



Exercise 2

For reproducibility, for each of the following tasks, set a fixed seed (seed=42).

- Generate a new dataset considering the characteristics defined below and plot both the dataset and the underlying model.

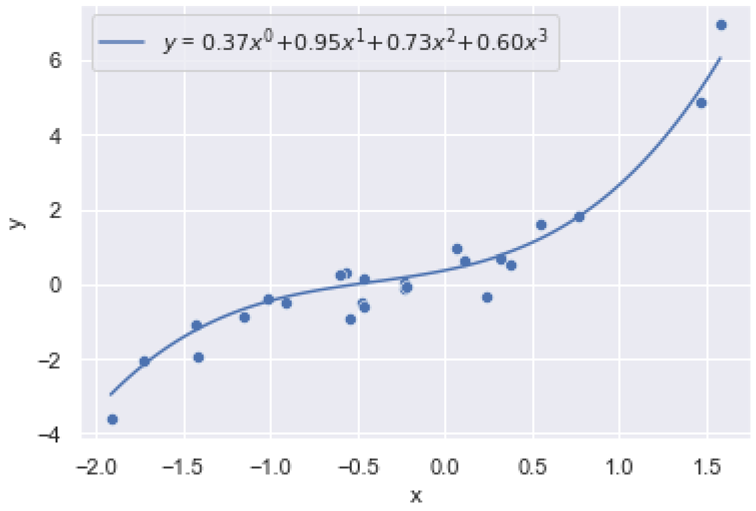
Parameter	Value (used in this notebook)	Description
`num_pairs`	25	amount of (x, y) pairs to generate
`variance`	0.5	variance within y w.r.t. defining function
`coefficients`	`np.random.rand(4)`	upper bound of random polynomial degree

Note: When defining the dataset, pass the random coefficients for the underlying model via the parameter `coefficients` . Also make sure to use the same random coefficients for plotting.

- Define a model by 'automatically' retrieving the optimal coefficients which minimize the MSE. Then, print out the optimal coefficients and plot both the data pairs and the optimized model.
- Compute and print out the difference between the random coefficients from the underlying model (used to generate the dataset) and the optimal coefficients ('automatically' computed) which minimize the MSE.

2.1. Create the dataset with random coefficients. Plot data pairs as well as the defined linear model.

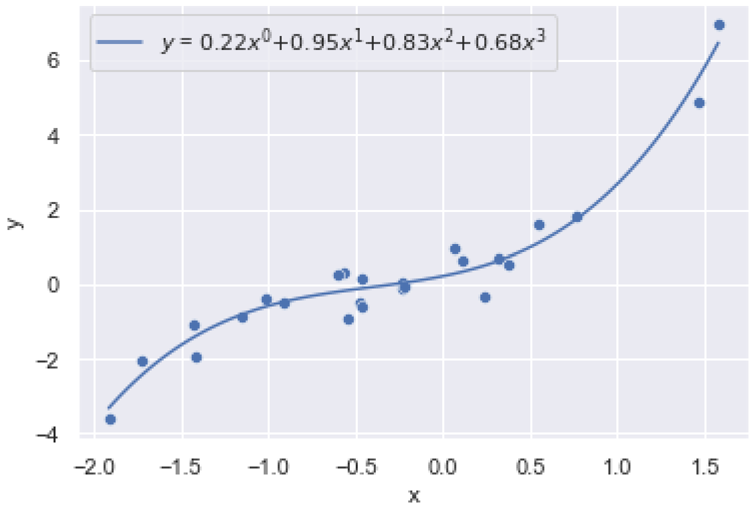
In [6]:
```
u4.set_seed(seed=42)

rand_coefficients = np.random.rand(4)

dataset = u4.get_dataset(num_pairs=25, variance=0.5,
                         coefficients=rand_coefficients)

u4.plot_model(dataset=dataset, coefficients=rand_coefficients)
```



2.2. Determine the coefficients 'automatically' and print them. Plot data pairs as well as the defined linear model.

```
In [7]: # Used 3 Degree Polynomial to be able to calculate the difference with the random
        mse_coefficients = u4.minimize_mse(dataset=dataset, degree=3)

        print(mse_coefficients)

        u4.plot_model(dataset=dataset, coefficients=mse_coefficients)
```

```
[0.22, 0.95, 0.83, 0.68]
```



2.3. Determine the difference between the random coefficients and the optimized coefficients and print this difference for each coefficient.

```
In [8]: diff_coefficients = rand_coefficients - mse_coefficients
        print(diff_coefficients)
```

```
[ 0.15454012  0.00071431 -0.09800606 -0.08134152]
```

Exercise 3

In this exercise, we will consider that y is a binary variable, i.e., the label of each data point is either 0 or 1. Under this premise, we will create a new dataset.

- Following the lecture notebook, choose the correct function from `u4_utils.py` to generate a dataset with binary labels consisting of 25 data pairs (feature, label). Then, plot it. For reproducibility, choose a fixed random seed of seed=27.
- Use the corresponding function in `u4_utils.py` to minimize the cross entropy loss and choose the combination of hyperparameters that enables the logistic regression model to best separate the two classes. Then, plot the dataset and the logistic regression model. Retrieve the optimal combination by taking the following values for each hyperparameter into account:

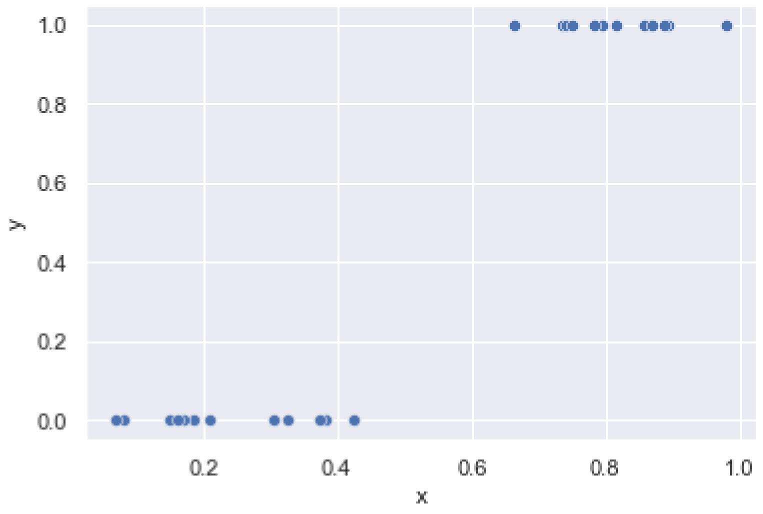
Hyperparameter	Values
`iterations`	10, 100

Hyperparameter	Values
`learning_rate`	1, 10, 100
`momentum`	0.3, 0.9

Note: For determining the best hyperparameters in this example, it is enough to simply look at the resulting model plot, i.e., you do not have to make any computations, a visual check is sufficient.

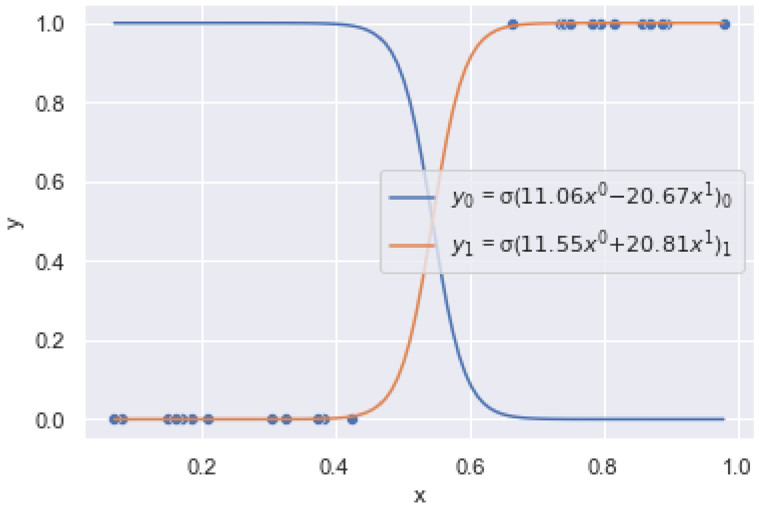
3.1. Create a logistic dataset and plot it.

In [9]:
```
u4.set_seed(seed=27)

dataset = u4.get_dataset_logistic(num_pairs=25)
sns.scatterplot(data=dataset, x="x", y="y");
```



3.2. Minimize cross entropy loss using the best hyperparameters. Plot data pairs as well as the defined logisitc model.

In [10]:
```python
ce_coefficients = u4.minimize_ce(
    dataset=dataset,
    iterations=100,
    learning_rate=10,
    momentum=0.9
)

u4.plot_logistic_model(
    dataset=dataset,
    coefficients=ce_coefficients
)
```



Exercise 4

Considering again a binary classification problem, we will now workwith the `DataSet_LR_a.csv` (same as in the lecture notebook).

- Load and process the dataset as shown in the lecture notebook but this time considering 75% (parameter `frac`) of the samples for training and 25% for testing. For reproducibility, set a seed=42.
- Considering `iterations=1000` and `momentum=0.9` and by using the function to minimize the cross entropy loss, find a learning rate that enables the logistic regression model to achieve an accuracy on the **test set** higher than 93.5%. Also print out the model's accuracy. For reproducibility, set a seed=42.
- Plot the 2D test dataset, once showing the true, actual labels (ground truth) and once showing the predicted labels.

4.1. Load the CSV file and create the training and test datasets.

In [11]:
```
u4.set_seed(seed=42)

dataset = u4.get_dataset_from_csv(path='resources/DataSet_LR_a.csv')

dataset_train = dataset.sample(frac=0.75, replace=False, axis=0)
dataset_test = dataset.drop(dataset_train.index)
```

4.2. Minimize cross entropy loss and identify a learning rate that achieves at least 93.5% accuracy on the test set. Print the model's accuracy.

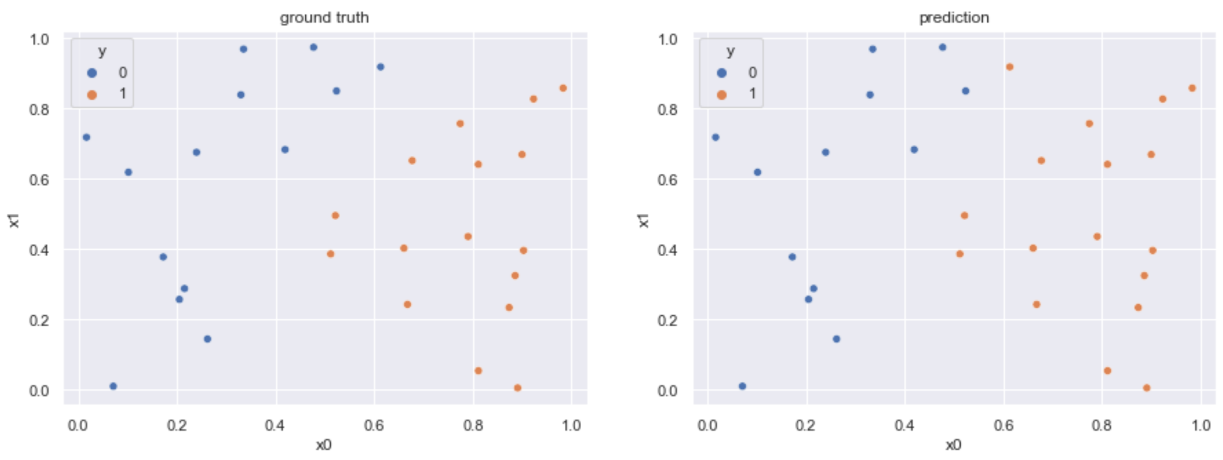
In [12]:
```
u4.set_seed(seed=42)

coefficients = u4.minimize_ce(dataset=dataset_train,
                              iterations=1000,
                              learning_rate=int(141),
                              momentum=0.9)
test_predictions = u4.predict_logistic(dataset_test.drop(columns="y"), coefficier
accuracy_test = (test_predictions == dataset_test["y"]).mean()
print(f"Original Accuracy on test set: {accuracy_test:.4f}")

import time

lr = pd.DataFrame(np.around([[l,0] for l in np.arange(1, 1001, 1)], 3),
                  columns=['learning_rate', 'test_accuracy'])
max_accuracy = 0

for i in lr.index:
    u4.set_seed(seed=42)
    stime = time.time()

    coefficients = u4.minimize_ce(dataset=dataset_train, iterations=1000,
                                  learning_rate=float(lr['learning_rate'].iloc[i]
                                  momentum=0.9)
    test_predictions = u4.predict_logistic(dataset_test.drop(columns="y"),
                                           coefficients)
    accuracy_test = (test_predictions == dataset_test["y"]).mean()

    lr['test_accuracy'].iloc[i] = accuracy_test
    if accuracy_test > max_accuracy:
        print('i', i,'Current Max. Accuracy:', f'{accuracy_test:.4f}',
              '| Learning Rate:',lr.learning_rate.iloc[i],
              '| Time Elapsed:', f'{time.time()-stime:.2f}')
        max_accuracy = accuracy_test

print(lr)
print(max(lr['test_accuracy']))
lr.to_csv('results_4.2.csv')
```

```
Original Accuracy on test set: 0.9333
i 0 Current Max. Accuracy: 0.9333 | Learning Rate: 1 | Time Elapsed: 0.28
i 113 Current Max. Accuracy: 0.9667 | Learning Rate: 114 | Time Elapsed: 0.30
     learning_rate  test_accuracy
0                1       0.933333
1                2       0.933333
2                3       0.933333
3                4       0.933333
4                5       0.933333
..             ...            ...
995            996       0.900000
996            997       0.900000
997            998       0.933333
998            999       0.900000
999           1000       0.933333

[1000 rows x 2 columns]
0.9666666666666667
```

```
**4.3. Plot the test data, once showing the ground truth and once showing the
predicted labels as 2D plots.**
```

In [13]:
```
u4.set_seed(seed=42)

coefficients = u4.minimize_ce(dataset=dataset_train,
                              iterations=1000,
                              learning_rate=int(114),
                              momentum=0.9)
test_predictions = u4.predict_logistic(dataset_test.drop(columns="y"),
                                       coefficients)
accuracy_test = (test_predictions == dataset_test["y"]).mean()

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15, 5))

axes[0].set_title("ground truth")
sns.scatterplot(data=dataset_test, x="x0", y="x1", hue="y", ax=axes[0])
axes[1].set_title("prediction")
sns.scatterplot(data=dataset_test.assign(y=test_predictions),
                x="x0", y="x1", hue="y", ax=axes[1])
```

Out[13]:
```
<AxesSubplot:title={'center':'prediction'}, xlabel='x0', ylabel='x1'>
```



Exercise 5

Considering the framework PyTorch and the instructions given in the lecture notebook, perform the following tasks:

- Generate a tensor of 2 dimensions with a shape of 2 x 4, containing random numbers from a standard normal distribution with (mean=0 and stddev=variance=1). Print out the tensor and its shape to be sure that your answer is correct. For reproducibility, set seed=42.
- Implement the function $g(x) = 5 \cdot x^2$ related to the previously generated tensor using PyTorch.
- Print the output of the function considering as input the tensor you have created in the first task of this exercise.
- Compute the gradient of the previously defined function $g(x)$ for $x = 0.5$ using the automatic differentiation functionality of PyTorch. Following the lecture notebook, print x as a tensor, the output of the function $g(x)$, and the gradient $g'(x)$

5.1. Create a random 2 x 4 tensor and print it and its shape.

```
In [14]: u4.set_seed(seed=42)
         x = torch.randn(2,2,4)
         print('X:\t', x)
         print('X Shape:\t', x.shape)
```

```
X:       tensor([[[ 1.9269,  1.4873,  0.9007, -2.1055],
         [ 0.6784, -1.2345, -0.0431, -1.6047]],

        [[-0.7521,  1.6487, -0.3925, -1.4036],
         [-0.7279, -0.5594, -0.7688,  0.7624]]])
X Shape:         torch.Size([2, 2, 4])
```

5.2. Define the function `g(x)` , call it with your tensor from above and print the result.

```
In [15]: def g(x):
              return 5 * (x**2)

         md_result = g(x)
         print('g(x):\n\t', md_result)
```

```
g(x):
         tensor([[[1.8565e+01, 1.1060e+01, 4.0565e+00, 2.2166e+01],
         [2.3013e+00, 7.6205e+00, 9.2740e-03, 1.2875e+01]],

        [[2.8285e+00, 1.3591e+01, 7.7020e-01, 9.8506e+00],
         [2.6491e+00, 1.5648e+00, 2.9556e+00, 2.9066e+00]]])
```

5.3. Create a scalar tensor with the value 0.5 and print it. Call the function `g(x)` from above using this tensor and print the result. Compute the gradient `g'(x)` using this tensor and print the result.

```
In [16]: x_scalar = torch.tensor([0.5], requires_grad=True )
         print('x_scalar:\t', x_scalar.item())

         y_scalar = g(x_scalar)
         print("g(x_scalar):\t", y_scalar.item())

         gradient = torch.autograd.grad(y_scalar, x_scalar)
         print("g'(x_scalar):\t", gradient)
```

```
x_scalar:        0.5
g(x_scalar):     1.25
g'(x_scalar):    (tensor([5.]),)
```

Exercise 6

With the MNIST dataset, perform the following tasks:

- Set the following hyperparameters (hint: `epochs` is the same as `iterations`):

Hyperparameter	Value
`batch_size`	9
`epochs`	1
`learning_rate`	0.01
`momentum`	0.7

- Load the dataset and fetch the test samples as well as test targets as shown in the lecture notebook.

Note: For reproducibility, for this one and **all** following tasks, set a fixed seed (seed=27).

- Train a logistic regression model on the training data, evaluate its accuracy on the test data and print out the results.
- Keeping `batch_size=9` and `epochs=1` , optimize the other two hyperparameters in a way that you achieve a better performance. Then, print out the results.

Note: You might consider some of the concepts learned in the previous units to automatize this process. Do not forget to set the seed if you want reproducible and comparable results.

6.1. Set up hyperparameters.

In [17]:
```
from types import SimpleNamespace

u4.set_seed(seed=27)
hyperparameters = SimpleNamespace(batch_size=9, epochs=1, learning_rate=0.01, mon
```

6.2. Get the data loaders for the training and test MNIST data. Fetch the samples and targets.

In [18]:
```
u4.set_seed(seed=27)
train_loader, test_loader = u4.get_dataset_mnist(batch_size=hyperparameters.batch

train_samples = torch.stack([x for x, y in train_loader.dataset]).flatten(start_d
train_targets = train_loader.dataset.targets

test_samples = torch.stack([x for x, y in test_loader.dataset]).flatten(start_dim
test_targets = test_loader.dataset.targets
```

6.3. Minimize the cross entropy loss using the training data to get the coefficients of the logisitc model. Get the predictions for the test data using this model and print the accuracy on the test set.

```
In [19]: u4.set_seed(seed=27)
         coefficients = u4.minimize_ce(dataset=train_loader,
                                       iterations=hyperparameters.epochs,
                                       learning_rate=hyperparameters.learning_rate,
                                       momentum=hyperparameters.momentum)

         test_predictions = u4.predict_logistic(test_samples, coefficients)

         accuracy_test = (test_predictions == test_targets.numpy()).mean()
         print(f"Accuracy on test set: {accuracy_test:.4f}")
```

```
Accuracy on test set: 0.9145
```

6.4. Optimize the hyperparameters `learning_rate` and/or `momentum` to get a higher accuracy than above.

```
In [21]: import time

u4.set_seed(seed=27)


min_lr = 0.01
max_lr = 1
step_lr = 0.01

min_mm = 0.1
max_mm = 1
step_mm = 0.1

lm = pd.DataFrame(np.around([[l,m,0] for l in np.arange(min_lr, max_lr+step_lr, s
                             for m in np.arange(min_mm, max_mm+step_mm,step_mm)],
                  columns=['learning_rate', 'momentum', 'test_accuracy'])

print('Iterations to find Optimum learning_rate & momentum: ', len(lm))


max_accuracy = 0

for i in lm.index:
    u4.set_seed(seed=27)
    stime = time.time()


    hyperparameters = SimpleNamespace(batch_size=9,
                                      epochs=1,
                                      learning_rate=float(lm['learning_rate'].ilo
                                      momentum=float(lm['momentum'].iloc[i]))

    coefficients = u4.minimize_ce(dataset=train_loader,
                                  iterations=hyperparameters.epochs,
                                  learning_rate=hyperparameters.learning_rate,
                                  momentum=hyperparameters.momentum)

    test_predictions = u4.predict_logistic(test_samples, coefficients)

    accuracy_test = (test_predictions == test_targets.numpy()).mean()

    lm['test_accuracy'].iloc[i] = accuracy_test
    if accuracy_test > max_accuracy:
        print('i', i, 'Current Max. Accuracy:', f'{accuracy_test:.4f}',
              '| Learning Rate:', lm['learning_rate'].iloc[i],
              '| Momentum:', lm['momentum'].iloc[i],
              '| Time Elapsed:', f'{time.time()-stime:.2f}')
        max_accuracy = accuracy_test
    lm.to_csv(str(i)+'.csv')


lm.to_csv('results_6.4.csv')
print(max(lm['test_accuracy']))
```

```
Iterations to find Optimum learning_rate & momentum:  1000
i 0 Current Max. Accuracy: 0.9093 | Learning Rate: 0.01 | Momentum: 0.1 | Tim
```

```
e Elapsed: 25.14
i 1 Current Max. Accuracy: 0.9100 | Learning Rate: 0.01 | Momentum: 0.2 | Tim
e Elapsed: 28.01
i 2 Current Max. Accuracy: 0.9102 | Learning Rate: 0.01 | Momentum: 0.3 | Tim
e Elapsed: 26.13
i 3 Current Max. Accuracy: 0.9113 | Learning Rate: 0.01 | Momentum: 0.4 | Tim
e Elapsed: 25.54
i 4 Current Max. Accuracy: 0.9117 | Learning Rate: 0.01 | Momentum: 0.5 | Tim
e Elapsed: 25.22
i 5 Current Max. Accuracy: 0.9128 | Learning Rate: 0.01 | Momentum: 0.6 | Tim
e Elapsed: 24.91
i 6 Current Max. Accuracy: 0.9145 | Learning Rate: 0.01 | Momentum: 0.7 | Tim
e Elapsed: 25.08
i 7 Current Max. Accuracy: 0.9161 | Learning Rate: 0.01 | Momentum: 0.8 | Tim
e Elapsed: 25.14
i 23 Current Max. Accuracy: 0.9162 | Learning Rate: 0.03 | Momentum: 0.4 | Ti
me Elapsed: 24.84
i 94 Current Max. Accuracy: 0.9173 | Learning Rate: 0.1 | Momentum: 0.5 | Tim
e Elapsed: 24.38
0.9173
```

```
In [ ]:
```