

Entrega 7: Detección de spam

Aprendizaje Automatico y Big Data- Alejandro Barrachina Argudo

1. Apartado A

Siguiendo las instrucciones del enunciado, el código queda tal que:

```

1 import sklearn.svm as svm
2 import scipy.io as sio
3 from sklearn.model_selection import train_test_split
4 import numpy as np
5 import matplotlib.pyplot as plt
6 import os
7 from utils_p7 import email2TokenList, getVocabDict
8 import codecs
9 import SVM_Trainer
10 import Logic_Regression_Trainer
11 import nn_trainer
12 import pytorch_trainer
13 import Poly_trainer
14
15 plot_folder: str = 'memoria/images'
16
17
18 def load_data(file: str) -> tuple[np.ndarray, np.ndarray]:
19     """Loads the data from a .mat file
20
21     Args:
22         file (str): name of the file
23
24     Returns:
25         tuple[np.ndarray, np.ndarray]: X and y data
26     """
27     data = sio.loadmat(file)
28     X = data['X']
29     y = data['y']
30     return X, y
31
32
33 def load_data3(file: str) -> tuple[np.ndarray, np.ndarray, np.ndarray, np.ndarray]:
34     """Loads the data from a .mat file
35
36     Args:
37         file (str): name of the file
38
39     Returns:
40         tuple[np.ndarray, np.ndarray, np.ndarray, np.ndarray]: X, y, Xval, yval data
41     """
42     data = sio.loadmat(file)
43     X = data['X']
44     y = data['y']
45     Xval = data['Xval']
46     yval = data['yval']
47     return X, y, Xval, yval
48
49
50 def kernel_linear(X: np.ndarray, y: np.ndarray, C: float) -> None:
51     """Linear kernel
52
53     Args:
54         X (np.ndarray): X train dataa
55         y (np.ndarray): y train data
56         C (float): regularization parameter
57     """
58     svm_lineal: svm.SVC = svm.SVC(kernel='linear', C=C)
59     svm_lineal.fit(X, y.ravel())
60     x1: np.ndarray = np.linspace(X[:, 0].min(), X[:, 0].max(), 100)
61     x2: np.ndarray = np.linspace(X[:, 1].min(), X[:, 1].max(), 100)
62     X1, X2 = np.meshgrid(x1, x2)
63     yp: np.ndarray = svm_lineal.predict(
64         np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape)
65     plt.contour(X1, X2, yp, colors='darkgreen', linewidths=1)

```

```

66 plt.scatter(X[y.ravel() == 1, 0], X[y.ravel() == 1, 1], c='b', marker='x')
67 plt.scatter(X[y.ravel() == 0, 0], X[y.ravel() == 0, 1], c='y', marker='o')
68 plt.xticks(np.arange(0, 5.5, 0.5))
69 plt.yticks(np.arange(1.5, 5.5, 0.5))
70 plt.savefig(f'{plot_folder}/SVM_lineal_c{C}.png', dpi=300)
71
72
73 def kerner_gaussiano(X: np.ndarray, y: np.ndarray, C: float, sigma: float) -> None:
74     """Gaussian kernel
75     Args:
76         X (np.ndarray): X train dataa
77         y (np.ndarray): y train data
78         C (float): regularization parameter
79         sigma (float): scale parameter
80     """
81     svm_gauss: svm.SVC = svm.SVC(kernel='rbf', C=C, gamma=1/(2*sigma**2))
82     svm_gauss.fit(X, y.ravel())
83     x1: np.ndarray = np.linspace(X[:, 0].min(), X[:, 0].max(), 100)
84     x2: np.ndarray = np.linspace(X[:, 1].min(), X[:, 1].max(), 100)
85     X1, X2 = np.meshgrid(x1, x2)
86     yp: np.ndarray = svm_gauss.predict(
87         np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape)
88     plt.contour(X1, X2, yp, colors='darkgreen', linewidths=1)
89     plt.scatter(X[y.ravel() == 1, 0], X[y.ravel() == 1, 1], c='b', marker='x')
90     plt.scatter(X[y.ravel() == 0, 0], X[y.ravel() == 0, 1], c='y', marker='o')
91     plt.xticks(np.arange(0.0, 1.2, 0.2))
92     plt.yticks(np.arange(0.4, 1.1, 0.1))
93     plt.savefig(f'{plot_folder}/SVM_gauss_c{C}_sigma{sigma}.png', dpi=300)
94
95
96 def seleccion_sigma_C() -> None:
97     """Selects the best C and sigma for the gaussian kernel
98     """
99     X, y, Xval, yval = load_data3('data/ex6data3.mat')
100     C_values: list[float] = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
101     sigma_values: list[float] = C_values
102     best_score: float = 0
103     best_params: tuple[float] = (0, 0)
104     for C in C_values:
105         for sigma in sigma_values:
106             svm_gauss = svm.SVC(kernel='rbf', C=C, gamma=1/(2*sigma**2))
107             svm_gauss.fit(X, y.ravel())
108             score = svm_gauss.score(Xval, yval)
109             if score > best_score:
110                 best_score = score
111                 best_params = (C, sigma)
112     print(f'Best score: {best_score}')
113     print(f'Best params: {best_params}')
114     svm_gauss: svm.SVC = svm.SVC(
115         kernel='rbf', C=best_params[0], gamma=1/(2*best_params[1]**2))
116     svm_gauss.fit(X, y.ravel())
117     x1: np.ndarray = np.linspace(X[:, 0].min(), X[:, 0].max(), 100)
118     x2: np.ndarray = np.linspace(X[:, 1].min(), X[:, 1].max(), 100)
119     X1, X2 = np.meshgrid(x1, x2)
120     yp: np.ndarray = svm_gauss.predict(
121         np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape)
122     plt.contour(X1, X2, yp, colors='darkgreen', linewidths=1)
123     plt.scatter(X[y.ravel() == 1, 0], X[y.ravel() == 1, 1], c='b', marker='x')
124     plt.scatter(X[y.ravel() == 0, 0], X[y.ravel() == 0, 1], c='y', marker='o')
125     plt.yticks(np.arange(-0.8, 0.7, 0.2))
126     plt.xticks(np.arange(-0.6, 0.4, 0.1))
127     plt.savefig(f'{plot_folder}/SVM_gauss_best.png', dpi=300)
128
129
130 def apartado_A() -> None:
131     """Apartado A
132     """
133     X, y = load_data('data/ex6data1.mat')
134     print("Linear kernel with C=1")
135     kernel_linear(X, y, 1.0)
136     plt.clf()
137     print("Linear kernel with C=100")
138     kernel_linear(X, y, 100.0)
139     X, y = load_data('data/ex6data2.mat')
140     plt.clf()
141     print("Gaussian kernel with C=1 and sigma=0.1")

```

```

142 kerner_gaussiano(X, y, 1.0, 0.1)
143 plt.clf()
144 print("Selecting C and sigma for gaussian kernel")
145 seleccion_sigma_C()
146
147
148 def load_data_spam() -> list[tuple[list[str], int]]:
149     """Loads the spam data
150     """
151     modes: list[str] = ['spam', 'easy_ham', 'hard_ham']
152     cantidades: list[int] = [500, 2551, 250]
153     spam_flag = [1, 0, 0]
154     correos = []
155     for mode, number, spam in zip(modes, cantidades, spam_flag):
156         progress = 0
157         length = 50
158         for file in range(1, number + 1):
159             file = str(file)
160             with codecs.open(f'./data_spam/spam/{mode}/{file.zfill(4)}.txt', 'r', encoding=
161 'utf-8', errors='ignore') as f:
162                 progress += 1
163                 bar_length = int(length * progress / number)
164                 bar = '[' + '=' * bar_length + \
165                     ' ' * (length - bar_length) + ']'
166                 print(f'\rLoading {mode} {bar} {progress}/{number}', end='')
167                 email = f.read()
168                 token_list = email2TokenList(email)
169                 correos.append((token_list, spam))
170         print()
171     print(len(correos))
172     return correos
173
174 def transform_mail(correos, vocab) -> tuple[np.ndarray, np.ndarray]:
175     """Transforms the emails into a matrix of length of the vocabulary with 1 if the word
176     is in the email
177
178     Args:
179         correos (_type_): mails
180         vocab (_type_): dictionary
181
182     Returns:
183         tuple[np.ndarray, np.ndarray]: transformed emails with label indicating if its spam
184         or not
185     """
186     X = []
187     y = []
188
189     for c, s in correos:
190         x = np.zeros(len(vocab) + 1)
191         for word in c:
192             if word in vocab:
193                 x[vocab[word]] = 1
194         X.append(x)
195         y.append(s)
196
197     return np.array(X), np.array(y)
198
199 def plot_results(train_scores: list[float], cv_scores: list[float], test_scores: list[float]
200 ], times: list[float]) -> None:
201     """Plots the results
202
203     Args:
204         train_scores (list[float]): train scores
205         cv_scores (list[float]): cv scores
206         test_scores (list[float]): test scores
207         times (list[float]): times
208     """
209     plt.clf()
210     X: np.ndarray = np.array(
211         ['Logistic Regression', 'SVM', 'NN', 'Pytorch', 'Poly'])
212     x = np.arange(len(X))
213     plt.bar(x-0.2,
214             train_scores, 0.2, label=f'Train')
215     plt.bar(x,
216             cv_scores, 0.2, label=f'CV')

```

```

214     plt.bar(x+0.2,
215             test_scores, 0.2, label=f'Test')
216     plt.legend()
217     plt.xticks(x, X)
218     plt.savefig(f'{plot_folder}/results.png', dpi=300)
219     plt.clf()
220     plt.plot(X, times)
221     plt.savefig(f'{plot_folder}/times.png', dpi=300)
222
223
224 def compare_results() -> None:
225     """Compares the results of the different models
226     """
227     lr_data = sio.loadmat('res/logistic_regression.mat')
228     svm_data = sio.loadmat('res/svm.mat')
229     nn_data = sio.loadmat('res/nn.mat')
230     pytorch_data = sio.loadmat('res/pytorch.mat')
231     poly_data = sio.loadmat('res/poly.mat')
232     print('Logistic Regression')
233     print(f'Score: {lr_data['train_score']}')
234     print(f'CV Score: {lr_data['cv_score']}')
235     print(f'Test Score: {lr_data['test_score']}')
236     print(f'Time: {lr_data['time']}')
237     print(f'Best params: {lr_data["best_params"]}')
238     print('SVM')
239     print(f'Score: {svm_data['train_score']}')
240     print(f'CV Score: {svm_data['cv_score']}')
241     print(f'Test Score: {svm_data['test_score']}')
242     print(f'Time: {svm_data['time']}')
243     print(f'Best params: {svm_data["best_params"]}')
244     print('NN')
245     print(f'Score: {nn_data['train_score']}')
246     print(f'CV Score: {nn_data['cv_score']}')
247     print(f'Test Score: {nn_data['test_score']}')
248     print(f'Time: {nn_data['time']}')
249     print(f'Best params: {nn_data["best_params"]}')
250     print('Pytorch')
251     print(f'Score: {pytorch_data['train_score']}')
252     print(f'CV Score: {pytorch_data['cv_score']}')
253     print(f'Test Score: {pytorch_data['test_score']}')
254     print(f'Time: {pytorch_data['time']}')
255     print(f'Best params: {pytorch_data["best_params"]}')
256     print('Poly')
257     print(f'Score: {poly_data['train_score']}')
258     print(f'CV Score: {poly_data['cv_score']}')
259     print(f'Test Score: {poly_data['test_score']}')
260     print(f'Time: {poly_data['time']}')
261     print(f'Best params: {poly_data["best_params"]}')
262
263     train_scores = [lr_data['train_score'][0][0], svm_data['train_score']
264                     [0][0], nn_data['train_score'][0][0], pytorch_data['train_score']
265                     [0][0], poly_data['train_score'][0][0]]
266     cv_scores = [lr_data['cv_score'][0][0], svm_data['cv_score']
267                 [0][0], nn_data['cv_score'][0][0], pytorch_data['cv_score'][0][0],
268                 poly_data['cv_score'][0][0]]
269     test_scores = [lr_data['test_score'][0][0], svm_data['test_score']
270                   [0][0], nn_data['test_score'][0][0], pytorch_data['test_score'][0][0],
271                   poly_data['test_score'][0][0]]
272     times = [lr_data['time'][0][0], svm_data['time'][0][0],
273              nn_data['time'][0][0], pytorch_data['time'][0][0], poly_data['time'][0][0]]
274     print(train_scores)
275     plot_results(train_scores, cv_scores, test_scores, times)
276
277
278 def apartado_B():
279     """Apartado B
280     """
281     correos = load_data_spam()
282     vocab = getVocabDict()
283     X, y = transform_mail(correos, vocab)
284     if not os.path.exists(f'res/svm.mat'):
285         print('Training SVM')
286         SVM_Trainer.trainer(X, y)
287     if not os.path.exists(f'res/logistic_regression.mat'):
288         print('Training Logistic Regression')
289         Logic_Regression_Trainer.LR_trainer(X, y)

```

```

287     if not os.path.exists(f'res/pytorch.mat'):
288         print('Training Pytorch')
289         pytorch_trainer.trainer(X, y)
290     if not os.path.exists(f'res/nn.mat'):
291         print('Training NN')
292         nn_trainer.trainer(X, y)
293     if not os.path.exists(f'res/poly.mat'):
294         print('Training Poly')
295         Poly_trainer.trainer(X, y)
296
297     compare_results()
298
299
300 def main() -> None:
301     apartado_A()
302     apartado_B()
303
304
305 if __name__ == '__main__':
306     main()

```

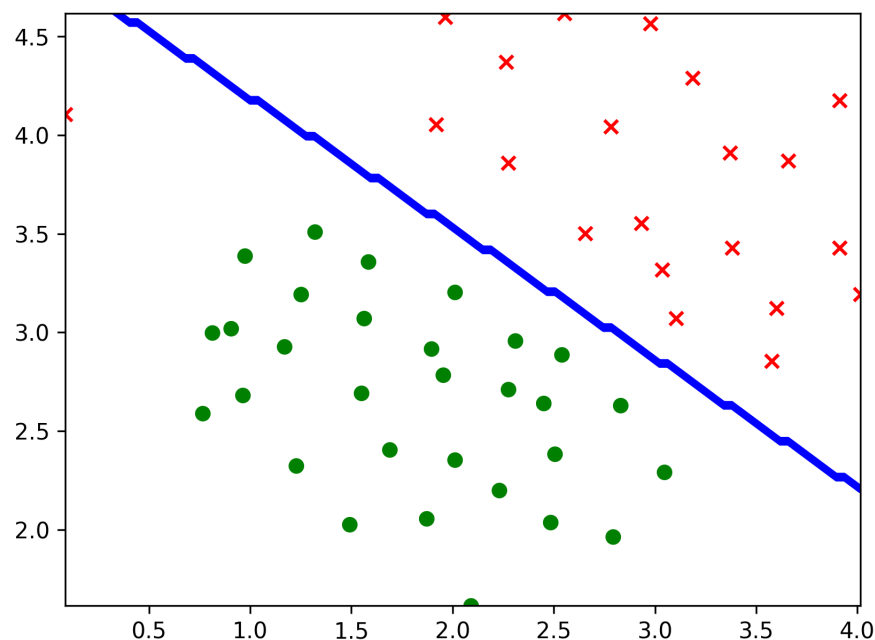
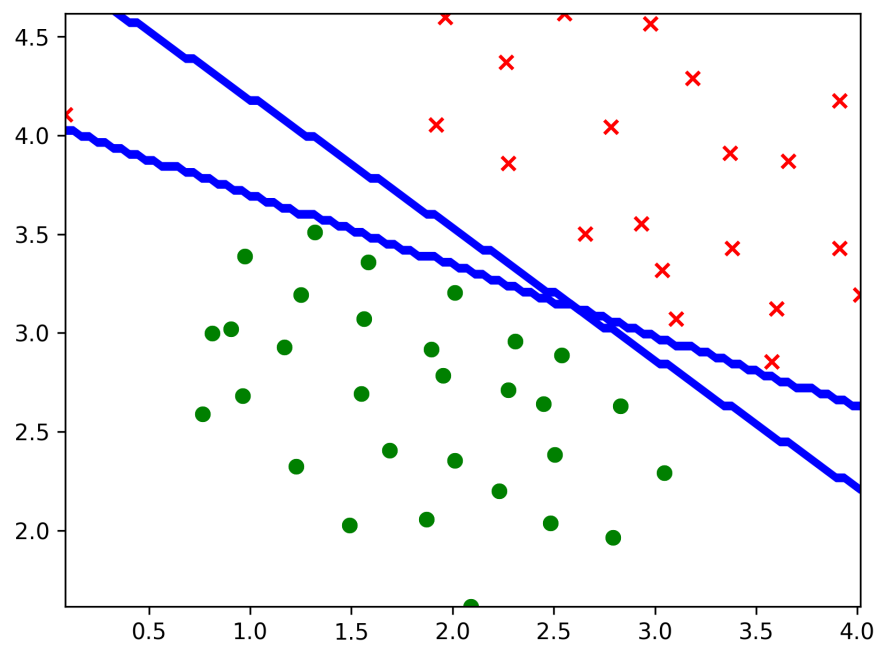
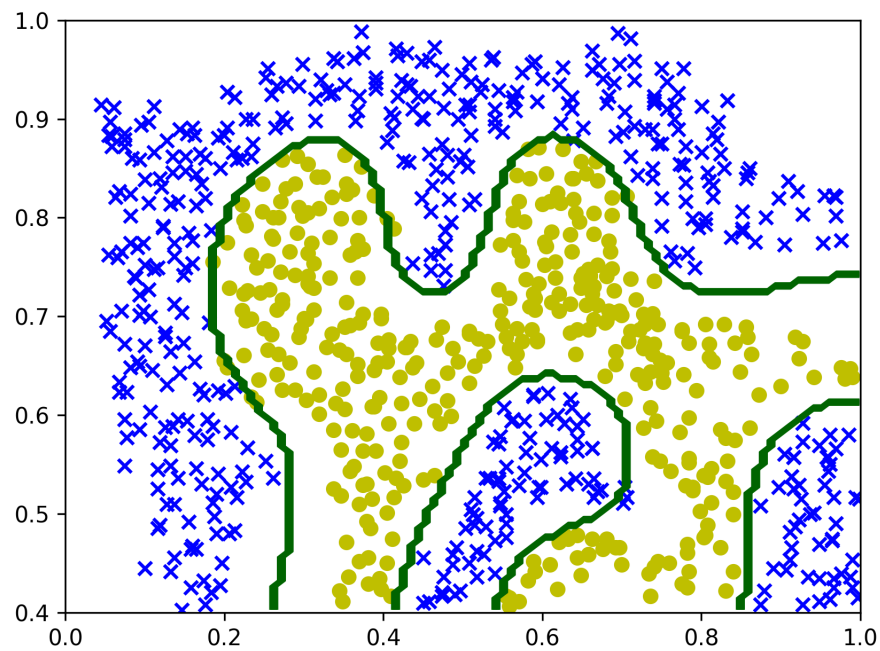


Figura 1.1: SVM lineal con $C=1.0$

Figura 1.2: SVM lineal con $C=100.0$ Figura 1.3: SVM gaussiano con $C=1.0$ y $\sigma=0.1$

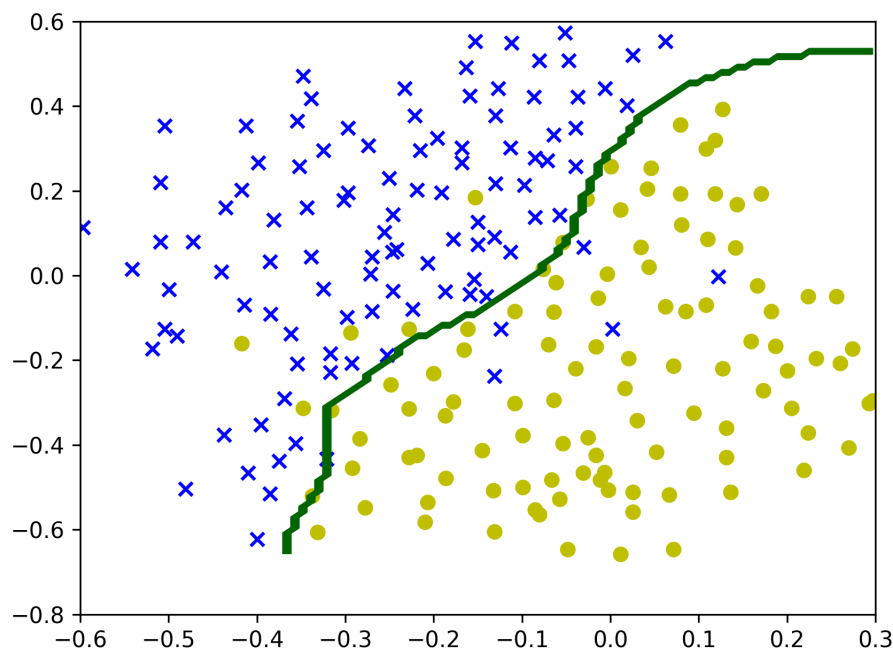


Figura 1.4: SVM gaussiano con $C=1.0$ y $\sigma=0.1$, mejor configuración para este problema

2. Apartado B

Para este problema usaremos distintos modelos:

- **Regresión lógica:** parece que sobreentrena en train, tiene como resultados 100, 97, 98 en train, validación y test respectivamente. Tarda 7.73 segundos en su mejor modelo con parámetros (10, 0.1)
- **SVM gaussiano:** mejor modelo de todos, tiene 98, 97, 98 en train, validación y test respectivamente. Tarda 1 segundos en su mejor modelo con parámetros (1.0, 10.)
- **NN:** el modelo que más tarda de todos (posiblemente porque está implementado en python a mano y no con una biblioteca hecha en un lenguaje competente) con resultados 96, 96 y 95 en train, validación y test respectivamente. Tarda 214 segundos en su mejor modelo con parámetros (3, 30)
- **Pytorch:** modelo entrenado en GPU con resultados 97, 97, 96 en train, validación y test respectivamente. Tarda 33 segundos en su mejor modelo con parámetros (0.001, 0.01)
- **PolynomialTransformer:** invariable a partir de grado 1, crashea el ordenador porque la matriz de datos es demasiado grande.

Código del entrenador de regresión lógica:

```

1 import numpy as np
2 import copy
3 import time
4 import scipy.io as sio
5 import concurrent.futures
6 from sklearn.model_selection import train_test_split
7
8
9 def compute_cost_reg(X: np.ndarray, y: np.ndarray, w: np.ndarray, b: float, lambda_: float
10 = 1) -> float:
11     """
12     Computes the cost over all examples
13     Args:
14         X : (array_like Shape (m,n)) data, m examples by n features
15         y : (array_like Shape (m,)) target value
16         w : (array_like Shape (n,)) Values of parameters of the model
17         b : (array_like Shape (n,)) Values of bias parameter of the model

```

```

17     lambda_ : (scalar, float)    Controls amount of regularization
18 Returns:
19     total_cost: (scalar)        cost
20 """
21
22     total_cost = compute_cost(X, y, w, b)
23     total_cost += (lambda_ / (2 * X.shape[0])) * np.sum(w**2)
24
25     return total_cost
26
27
28 def loss(X: np.ndarray, Y: np.ndarray, fun: np.ndarray, w: np.ndarray, b: float) -> float:
29     """loss function for the logistic regression
30
31     Args:
32         X (np.ndarray): X values
33         Y (np.ndarray): Expected y results
34         fun (np.ndarray): logistic regression function
35         w (np.ndarray): weights
36         b (float): bias
37
38     Returns:
39         float: total loss of the regression
40     """
41
42     return (-Y * np.log(fun(X, w, b) + 1e-6)) - ((1 - Y) * np.log(1 - fun(X, w, b) + 1e-6))
43
44
45 def compute_cost(X: np.ndarray, y: np.ndarray, w: np.ndarray, b: float, lambda_=None) ->
float:
46     """
47     Computes the cost over all examples
48     Args:
49         X : (ndarray Shape (m,n)) data, m examples by n features
50         y : (array_like Shape (m,)) target value
51         w : (array_like Shape (n,)) Values of parameters of the model
52         b : scalar Values of bias parameter of the model
53         lambda_: unused placeholder
54     Returns:
55         total_cost: (scalar)        cost
56     """
57     # apply the loss function for each element of the x and y arrays
58     loss_v = loss(X, y, function, w, b)
59     total_cost = np.sum(loss_v)
60     total_cost /= X.shape[0]
61
62     return total_cost
63
64
65 def compute_gradient(X: np.ndarray, y: np.ndarray, w: np.ndarray, b: float, lambda_=None)
-> tuple[float, np.ndarray]:
66     """
67     Computes the gradient for logistic regression
68
69     Args:
70         X : (ndarray Shape (m,n)) variable such as house size
71         y : (array_like Shape (m,1)) actual value
72         w : (array_like Shape (n,1)) values of parameters of the model
73         b : (scalar)                value of parameter of the model
74         lambda_: unused placeholder
75     Returns
76         dj_db: (scalar)                The gradient of the cost w.r.t. the parameter b.
77         dj_dw: (array_like Shape (n,1)) The gradient of the cost w.r.t. the parameters w.
78     """
79
80     func = function(X, w, b)
81
82     dj_dw = np.dot(func - y, X)
83     dj_dw /= X.shape[0]
84
85     dj_db = np.sum(func - y)
86     dj_db /= X.shape[0]
87
88     return dj_db, dj_dw
89
90

```



```

91 def compute_gradient_reg(X: np.ndarray, y: np.ndarray, w: np.ndarray, b: float, lambda_:
float = 1) -> tuple[float, np.ndarray]:
92     """
93     Computes the gradient for linear regression
94
95     Args:
96         X : (ndarray Shape (m,n))    variable such as house size
97         y : (ndarray Shape (m,))      actual value
98         w : (ndarray Shape (n,))      values of parameters of the model
99         b : (scalar)                  value of parameter of the model
100         lambda_ : (scalar,float)      regularization constant
101     Returns
102         dj_db: (scalar)                The gradient of the cost w.r.t. the parameter b.
103         dj_dw: (ndarray Shape (n,))    The gradient of the cost w.r.t. the parameters w.
104
105     """
106     dj_db, dj_dw = compute_gradient(X, y, w, b)
107     dj_dw += (lambda_ / X.shape[0]) * w
108
109     return dj_db, dj_dw
110
111
112 def gradient_descent(X: np.ndarray, y: np.ndarray, w_in: np.ndarray, b_in: float,
cost_function: float, gradient_function: float, alpha: float, num_iters: int, lambda_:
float = None) -> tuple[np.ndarray, float, np.ndarray, np.ndarray]:
113     """
114     Performs batch gradient descent to learn theta. Updates theta by taking
115     num_iters gradient steps with learning rate alpha
116
117     Args:
118         X : (array_like Shape (m, n))
119         y : (array_like Shape (m,))
120         w_in : (array_like Shape (n,))    Initial values of parameters of the model
121         b_in : (scalar)                    Initial value of parameter of the model
122         cost_function:                      function to compute cost
123         alpha : (float)                     Learning rate
124         num_iters : (int)                   number of iterations to run gradient descent
125         lambda_ (scalar, float)             regularization constant
126
127     Returns:
128         w : (array_like Shape (n,)) Updated values of parameters of the model after
129         running gradient descent
130         b : (scalar)                    Updated value of parameter of the model after
131         running gradient descent
132         J_history : (ndarray): Shape (num_iters,) J at each iteration,
133         primarily for graphing later
134     """
135
136     w = copy.deepcopy(w_in)
137     b = b_in
138     predict_history = [predict_check(X, y, w, b)]
139     J_history = [cost_function(X, y, w, b, lambda_)]
140
141     for i in range(num_iters):
142         dj_db, dj_dw = gradient_function(X, y, w, b, lambda_)
143         w = w - (alpha * dj_dw)
144         b -= alpha * dj_db
145         J_history.append(cost_function(X, y, w, b, lambda_))
146         predict_history.append(predict_check(X, y, w, b))
147
148     return w, b, np.array(J_history), predict_history
149
150
151 def predict(X, w, b) -> np.ndarray:
152     """
153     Predict whether the label is 0 or 1 using learned logistic
154     regression parameters w and b
155
156     Args:
157         X : (ndarray Shape (m, n))
158         w : (array_like Shape (n,))    Parameters of the model
159         b : (scalar, float)             Parameter of the model
160
161     Returns:
162         p: (ndarray (m,1))
163         The predictions for X using a threshold at 0.5

```

```

164     """
165
166     p = np.vectorize(lambda x: 1 if x > 0.5 else 0)(
167         function(X, w, b))
168     return p
169
170
171 def predict_check(X, Z, w, b) -> float:
172     """Gives a percentage of the accuracy of the prediction
173
174     Args:
175         X (_type_): X train data
176         Z (_type_): expected values
177         w (_type_): weights
178         b (_type_): bias
179
180     Returns:
181         float: percentage of accuracy
182     """
183     p = predict(X, w, b)
184     return np.sum(p == Z) / Z.shape[0]
185
186
187 def train_model(X: np.ndarray, y: np.ndarray, x_cv: np.ndarray, y_cv: np.ndarray, alpha:
188     float, lambda_: float, num_iters: int) -> tuple[float, float, float]:
189     """Train the model with the given parameters
190
191     Args:
192         X (np.ndarray): Training data
193         y (np.ndarray): Training target
194         x_cv (np.ndarray): Cross validation data
195         y_cv (np.ndarray): Cross validation target
196         alpha (float): Learning rate
197         lambda_ (float): Regularization parameter
198         num_iters (int): Number of iterations
199
200     Returns:
201         tuple[float, float, float]: Learning rate, Regularization parameter, Score
202     """
203     print(f'Alpha: {alpha} Lambda: {lambda_}')
204     m, n = X.shape
205     X = np.hstack((np.ones((m, 1)), X))
206     x_cv = np.hstack((np.ones((x_cv.shape[0], 1)), x_cv))
207     w = np.zeros(X.shape[1])
208     b = 1
209     w, b, _, _ = gradient_descent(
210         X, y, w, b, compute_cost_reg, compute_gradient_reg, alpha, num_iters, lambda_)
211     score = predict_check(x_cv, y_cv, w, b)
212     return (alpha, lambda_, score)
213
214
215 def LR_trainer(X: np.ndarray, y: np.ndarray) -> None:
216     """Trains the model with the given data
217
218     Args:
219         X (np.ndarray): Input data
220         y (np.ndarray): Target data
221     """
222     alphas = [0.1, 0.3, 1, 3, 10, 30]
223     lambdas = [0.1, 0.3, 1, 3, 10, 30]
224     num_iters = 1000
225     best_score = 0
226     best_params = (0, 0)
227
228     X_train, X_test, y_train, y_test = train_test_split(
229         X, y, test_size=0.3, shuffle=True, random_state=22)
230     X_cv, X_test, y_cv, y_test = train_test_split(
231         X_test, y_test, test_size=0.5, shuffle=True, random_state=22)
232
233     with concurrent.futures.ProcessPoolExecutor() as executor:
234         futures = []
235         for lambda_ in lambdas:
236             for alpha in alphas:
237                 futures.append(executor.submit(
238                     train_model, X_train, y_train, X_cv, y_cv, alpha, lambda_, num_iters))
239
240         for future in concurrent.futures.as_completed(futures):
241             alpha, lambda_, score = future.result()
242             print(f'Alpha: {alpha} Lambda: {lambda_} Score: {score}')

```

```

239         if score > best_score:
240             best_score = score
241             best_params = (alpha, lambda_)
242
243     print(f'Best score: {best_score}')
244     print(f'Best params: {best_params}')
245
246     start = time.time()
247     w = np.zeros(X.shape[1] + 1)
248     b = 1
249     X_train = np.hstack((np.ones((X_train.shape[0], 1)), X_train))
250     w, b, _, _ = gradient_descent(
251         X_train, y_train, w, b, compute_cost_reg, compute_gradient_reg, best_params[0],
252         num_iters, best_params[1])
253     end = time.time()
254     print(f'Training time: {end-start}')
255     train_score = predict_check(X_train, y_train, w, b)
256     print(f'Train score: {train_score}')
257     X_cv = np.hstack((np.ones((X_cv.shape[0], 1)), X_cv))
258     cv_score = predict_check(X_cv, y_cv, w, b)
259     print(f'CV score: {cv_score}')
260     X_test = np.hstack((np.ones((X_test.shape[0], 1)), X_test))
261     test_score = predict_check(X_test, y_test, w, b)
262     print(f'Test score: {test_score}')
263     sio.savemat('res/logistic_regression.mat', {'w': w, 'b': b, 'train_score': train_score,
264         'cv_score': cv_score, 'test_score': test_score, 'best_params': best_params,
265         'time': end-start})

```

Código del entrenador de SVM gaussiano:

```

1  import numpy as np
2  import matplotlib.pyplot as plt
3  from sklearn.model_selection import train_test_split
4  import time
5  from sklearn import svm
6  import scipy.io as sio
7  import concurrent.futures
8
9
10 def train_model(C: float, sigma: float, x_train: np.ndarray, y_train: np.ndarray, x_cv: np.
11     ndarray, y_cv: np.ndarray) -> tuple[float, float, float]:
12     """Train the model with the given parameters
13     Args:
14         C (float): Regularization parameter
15         sigma (float): Gaussian kernel parameter
16         x_train (np.ndarray): Training data
17         y_train (np.ndarray): Training target
18         x_cv (np.ndarray): Cross validation data
19         y_cv (np.ndarray): Cross validation target
20     Returns:
21         tuple[float, float, float]: Regularization parameter, Gaussian kernel parameter,
22         Score
23     """
24     print(f'C: {C} sigma: {sigma}')
25     svm_gauss = svm.SVC(kernel='rbf', C=C, gamma=1/(2*sigma**2))
26     svm_gauss.fit(x_train, y_train.ravel())
27     score = svm_gauss.score(x_cv, y_cv.ravel())
28     return (C, sigma, score)
29
30 def trainer(X: np.ndarray, y: np.ndarray) -> None:
31     """Trains the model with the given data
32     Args:
33         X (np.ndarray): Input data
34         y (np.ndarray): Target data
35     """
36     C_values = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
37     sigma_values = C_values
38     x_train, x_test, y_train, y_test = train_test_split(
39         X, y, test_size=0.3, random_state=22)
40     x_cv, x_test, y_cv, y_test = train_test_split(
41         x_test, y_test, test_size=0.5, random_state=22)
42     best_score = 0
43     best_params = (0, 0)
44
45     with concurrent.futures.ProcessPoolExecutor() as executor:

```

```

45     futures = []
46     for C in C_values:
47         for sigma in sigma_values:
48             futures.append(executor.submit(
49                 train_model, C, sigma, x_train, y_train, x_cv, y_cv))
50
51     for future in concurrent.futures.as_completed(futures):
52         C, sigma, score = future.result()
53         print(f'C: {C} sigma: {sigma} score: {score}')
54         if score > best_score:
55             best_score = score
56             best_params = (C, sigma)
57
58     print(f'Best score: {best_score}')
59
60     start = time.time()
61     svm_gauss = svm.SVC(
62         kernel='rbf', C=best_params[0], gamma=1/(2*best_params[1]**2))
63     svm_gauss.fit(x_train, y_train.ravel())
64     end = time.time()
65     print(f'Training time: {end-start}')
66
67     test_score = svm_gauss.score(x_test, y_test)
68     cv_score = svm_gauss.score(x_cv, y_cv)
69     train_score = svm_gauss.score(x_train, y_train)
70     sio.savemat('res/svm.mat', {'train_score': train_score,
71                                'cv_score': cv_score, 'test_score': test_score,
72                                'best_params': best_params, 'time': end-start})

```

Código del entrenador de NN:

```

1  import numpy as np
2  import scipy.io as sio
3  import time
4  from sklearn.model_selection import train_test_split
5
6
7  def sigmoid(z: np.ndarray) -> np.ndarray:
8      """
9      Compute the sigmoid of z
10
11      Args:
12          z (ndarray): A scalar, numpy array of any size.
13
14      Returns:
15          g (ndarray): sigmoid(z), with the same shape as z
16
17      """
18
19      g = 1/(1+np.exp(-z))
20
21      return g
22
23
24  def fix_data(X: np.ndarray) -> np.ndarray:
25      """Fixes the data to avoid log(0) errors
26
27      Args:
28          X (np.ndarray): train data
29
30      Returns:
31          np.ndarray: matrix with no 0 or 1 values
32      """
33      return X + 1e-7
34
35
36  def cost(theta1: np.ndarray, theta2: np.ndarray, X: np.ndarray, y: np.ndarray, lambda_:
37           float = 0.0) -> float:
38      """
39      Compute cost for 2-layer neural network.
40
41      Parameters
42      -----
43      theta1 : array_like
44          Weights for the first layer in the neural network.
45          It has shape (2nd hidden layer size x input size + 1)

```

```

45
46     theta2: array_like
47         Weights for the second layer in the neural network.
48         It has shape (output layer size x 2nd hidden layer size + 1)
49
50     X : array_like
51         The inputs having shape (number of examples x number of dimensions).
52
53     y : array_like
54         1-hot encoding of labels for the input, having shape
55         (number of examples x number of labels).
56
57     lambda_ : float
58         The regularization parameter.
59
60     Returns
61     -----
62     J : float
63         The computed value for the cost function.
64
65     """
66     L = 2
67     layers = [theta1, theta2]
68     k: int = y.shape[1]
69     h, z = neural_network(X, [theta1, theta2])
70
71     h = h[-1]
72
73     h = fix_data(h)
74
75     J = y * np.log(h + 1e-7)
76     J += (1 - y) * np.log(1 - h + 1e-7)
77
78     J = -1 / X.shape[0] * np.sum(J)
79
80     if lambda_ != 0:
81         reg = 0
82         for layer in layers:
83             reg += np.sum(layer[:, 1:] ** 2)
84         J += lambda_ / (2 * X.shape[0]) * reg
85     return J
86
87
88 def neural_network(X: np.ndarray, thetas: np.ndarray) -> tuple[np.ndarray, np.ndarray]:
89     """Generate the neural network with a given set of weights
90
91     Args:
92         X (np.ndarray): data
93         thetas (np.ndarray): array containing the weights for each layer
94
95     Returns:
96         tuple[np.ndarray, np.ndarray]: tuple containing the activations and the z values
97         for each layer
98     """
99     a = []
100     z = []
101     a.append(X.copy())
102     for theta in thetas:
103         a[-1] = np.hstack((np.ones((a[-1].shape[0], 1)), a[-1]))
104         z.append(np.dot(a[-1], theta.T))
105         a.append(sigmoid(z[-1]))
106     return a, z
107
108 def backprop(theta1: np.ndarray, theta2: np.ndarray, X: np.ndarray, y: np.ndarray, lambda_:
109             float) -> tuple[float, np.ndarray, np.ndarray]:
110     """
111     Compute cost and gradient for 2-layer neural network.
112
113     Parameters
114     -----
115     theta1 : array_like
116         Weights for the first layer in the neural network.
117         It has shape (2nd hidden layer size x input size + 1)
118
119     theta2: array_like

```

```

119     Weights for the second layer in the neural network.
120     It has shape (output layer size x 2nd hidden layer size + 1)
121
122     X : array_like
123         The inputs having shape (number of examples x number of dimensions).
124
125     y : array_like
126         1-hot encoding of labels for the input, having shape
127         (number of examples x number of labels).
128
129     lambda_ : float
130         The regularization parameter.
131
132     Returns
133     -----
134     J : float
135         The computed value for the cost function.
136
137     grad1 : array_like
138         Gradient of the cost function with respect to weights
139         for the first layer in the neural network, theta1.
140         It has shape (2nd hidden layer size x input size + 1)
141
142     grad2 : array_like
143         Gradient of the cost function with respect to weights
144         for the second layer in the neural network, theta2.
145         It has shape (output layer size x 2nd hidden layer size + 1)
146
147     """
148     m = X.shape[0]
149     L = 2
150
151     delta = np.empty(2, dtype=object)
152     delta[0] = np.zeros(theta1.shape)
153     delta[1] = np.zeros(theta2.shape)
154
155     a, z = neural_network(X, [theta1, theta2])
156
157     for k in range(m):
158         a1k = a[0][k, :]
159         a2k = a[1][k, :]
160         hk = a[2][k, :]
161         yk = y[k, :]
162
163         d3k = hk - yk
164         d2k = np.dot(theta2.T, d3k) * a2k * (1 - a2k)
165
166         delta[0] = delta[0] + \
167             np.matmul(d2k[1:, np.newaxis], a1k[np.newaxis, :])
168         delta[1] = delta[1] + np.matmul(d3k[:, np.newaxis], a2k[np.newaxis, :])
169
170     grad1 = delta[0] / m
171     grad2 = delta[1] / m
172
173     if lambda_ != 0:
174         grad1[:, 1:] += lambda_ / m * theta1[:, 1:]
175         grad2[:, 1:] += lambda_ / m * theta2[:, 1:]
176
177     J = cost(theta1, theta2, X, y, lambda_)
178
179     return (J, grad1, grad2)
180
181
182 def gradient_descent(X: np.ndarray, y: np.ndarray, theta1: np.ndarray, theta2: np.ndarray,
183                     alpha: float, lambda_: float, num_iters: int) -> tuple[np.ndarray, np.ndarray, np.
184                                     ndarray]:
185     """Generates the gradient descent for the neural network
186
187     Args:
188         X (np.ndarray): Train data
189         y (np.ndarray): Expected output in one hot encoding
190         theta1 (np.ndarray): initial weights for the first layer
191         theta2 (np.ndarray): initial weights for the second layer
192         alpha (float): learning rate
193         lambda_ (float): regularization parameter
194         num_iters (int): number of iterations to run

```

```

193
194 Returns:
195     tuple[np.ndarray, np.ndarray, np.ndarray]: tuple with the final weights for the
196     first and second layer and the cost history
197     """
198     m = X.shape[0]
199     J_history = np.zeros(num_iters)
200     for i in range(num_iters):
201         print('Iteration: ', i + 1, '/', num_iters, end='\r')
202         J, grad1, grad2 = backprop(theta1, theta2, X, y, lambda_)
203         theta1 = theta1 - alpha * grad1
204         theta2 = theta2 - alpha * grad2
205         J_history[i] = J
206     print('Gradient descent finished.')
207     return theta1, theta2, J_history
208
209 def train_model(X, y, x_cv, y_cv, alpha, lambda_, num_iters):
210     start = time.time()
211     print(f'Alpha: {alpha} Lambda: {lambda_}')
212     input_layer_size = X.shape[1]
213     hidden_layer_size = 125
214     num_labels = 2
215     yA = [0 if i == 1 else 1 for i in y]
216     yB = [1 if i == 1 else 0 for i in y]
217     y_encoded = np.array([yA, yB]).T
218
219     theta1 = np.random.rand(hidden_layer_size, input_layer_size + 1)
220     theta2 = np.random.rand(num_labels, hidden_layer_size + 1)
221
222     theta1, theta2, J_history = gradient_descent(
223         X, y_encoded, theta1, theta2, alpha, lambda_, num_iters)
224
225     score = predict_percentage(x_cv, y_cv, theta1, theta2)
226     time = time.time() - start
227     return (alpha, lambda_, score, theta1, theta2)
228
229
230 def prediction(X: np.ndarray, theta1: np.ndarray, theta2: np.ndarray) -> np.ndarray:
231     """Generates the neural network prediction
232
233     Args:
234         X (np.ndarray): data
235         theta1 (np.ndarray): first layer weight
236         theta2 (np.ndarray): second layer weight
237
238     Returns:
239         np.ndarray: best prediction for each row in 'X'
240     """
241     m = X.shape[0]
242     p = np.zeros(m)
243     a, z = neural_network(X, [theta1, theta2])
244     h = a[-1]
245
246     return np.argmax(h, axis=1)
247
248
249 def predict_percentage(X: np.ndarray, y: np.ndarray, theta1: np.ndarray, theta2: np.ndarray) -> float:
250     """Gives the accuracy of the neural network
251
252     Args:
253         X (ndarray): Train data
254         y (ndarray): Expected output
255         theta1 (ndarray): First layer weights
256         theta2 (ndarray): Second layer weights
257
258     Returns:
259         float: Accuracy of the neural network
260     """
261     m = X.shape[0]
262     p = prediction(X, theta1, theta2)
263
264     return p[p == y].size / m
265
266

```

```

267 def trainer(X: np.ndarray, y: np.ndarray) -> None:
268     lambdas = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
269     alphas = lambdas
270     num_iters = 100
271     best_score = 0
272     best_params = (0, 0)
273     input_layer_size = X.shape[1]
274     hidden_layer_size = 125
275     num_labels = 2
276     best_time = 0
277     X_train, X_test, y_train, y_test = train_test_split(
278         X, y, test_size=0.3, random_state=22)
279     X_cv, X_test, y_cv, y_test = train_test_split(
280         X_test, y_test, test_size=0.5, random_state=22)
281     model = (np.array([]), np.array([]))
282
283     for alpha in alphas:
284         for lambda_ in lambdas:
285
286             start = time.time()
287             print(f'Alpha: {alpha} Lambda: {lambda_}')
288             input_layer_size = X.shape[1]
289             hidden_layer_size = 125
290             num_labels = 2
291             yA = [0 if i == 1 else 1 for i in y]
292             yB = [1 if i == 1 else 0 for i in y]
293             y_encoded = np.array([yA, yB]).T
294             theta1 = np.random.rand(hidden_layer_size, input_layer_size + 1)
295             theta2 = np.random.rand(num_labels, hidden_layer_size + 1)
296
297             theta1, theta2, J_history = gradient_descent(
298                 X, y_encoded, theta1, theta2, alpha, lambda_, num_iters)
299
300             score = predict_percentage(X_cv, y_cv, theta1, theta2)
301             print(f'Score: {score}')
302             aux_time = time.time() - start
303             if score > best_score:
304                 best_score = score
305                 best_params = (alpha, lambda_)
306                 model = (theta1, theta2)
307                 best_time = aux_time
308             print(f'Best score: {best_score}')
309             print(f'Best params: {best_params}')
310
311             theta1 = np.random.rand(hidden_layer_size, input_layer_size + 1)
312             theta2 = np.random.rand(num_labels, hidden_layer_size + 1)
313             yA = [0 if i == 1 else 1 for i in y_train]
314             yB = [1 if i == 1 else 0 for i in y_train]
315             y_encoded = np.array([yA, yB]).T
316
317             theta1, theta2, = model
318             print(f'Training time: {best_time}')
319
320             train_score = predict_percentage(X_train, y_train, theta1, theta2)
321             print(f'Train score: {train_score}')
322             cv_score = predict_percentage(X_cv, y_cv, theta1, theta2)
323             print(f'CV score: {cv_score}')
324             test_score = predict_percentage(X_test, y_test, theta1, theta2)
325             print(f'Test score: {test_score}')
326             sio.savemat('res/nn.mat',
327                 {'theta1': theta1, 'theta2': theta2, 'train_score': train_score, 'cv_score':
328                 : cv_score, 'test_score': test_score, 'best_params': best_params, 'time': best_time})

```

Código del entrenador de Pytorch:

```

1 import torch.nn as nn
2 import torch.optim as optim
3 import numpy as np
4 from sklearn.model_selection import train_test_split
5 import scipy.io as sio
6 import time
7 import torch
8
9 # Select cuda device if available to speed up training
10 device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
11

```



```

12 if torch.cuda.is_available():
13     print(f'Using GPU {torch.cuda.get_device_name()}')
14 else:
15     print('Using CPU')
16
17
18 def train_data(x: np.ndarray, y: np.ndarray) -> torch.utils.data.DataLoader:
19     """ Create a DataLoader object from the input data
20     Args:
21         x (np.ndarray): Input data
22         y (np.ndarray): Target data
23     """
24     return torch.utils.data.DataLoader(torch.utils.data.TensorDataset(
25         torch.tensor(x, dtype=torch.float).to(device), torch.tensor(y).to(device)),
26         batch_size=2, shuffle=True)
27
28 def train_model(model: nn.Sequential, train_dl: torch.utils.data.DataLoader, criterion: nn.
29     CrossEntropyLoss, optimizer: optim.Adam, epochs: int) -> nn.Sequential:
30     """ Train the model with the given data
31     Args:
32         model (nn.Sequential): Model to train
33         train_dl (torch.utils.data.DataLoader): DataLoader object with the training data
34         criterion (nn.CrossEntropyLoss): Loss function
35         optimizer (optim.Adam): Optimizer
36         epochs (int): Number of epochs to train the model
37     Returns:
38         nn.Sequential: Trained model
39     """
40     for epoch in range(epochs):
41         model.train()
42         for x, y in train_dl:
43             optimizer.zero_grad()
44             y_pred = model(x)
45             loss = criterion(y_pred, y)
46             loss.backward()
47             optimizer.step()
48             print(f'Epoch: {epoch}, Loss: {loss.item()}')
49     return model
50
51 def ComplexModel(input_size: int) -> nn.Sequential:
52     """Creates a Sequential model with 3 layers
53
54     Args:
55         input_size (int): input size of the model
56
57     Returns:
58         nn.Sequential: base model
59     """
60     return nn.Sequential(
61         nn.Linear(input_size, 512),
62         nn.ReLU(),
63         nn.Linear(512, 10),
64         nn.ReLU(),
65         nn.Linear(10, 2),
66         nn.Sigmoid()
67     ).to(device)
68
69
70 def pred_check(pred: torch.Tensor, y: np.ndarray) -> float:
71     """Gives the accuracy of the model in percentage
72
73     Args:
74         pred (torch.Tensor): predictions made by the model
75         y (np.ndarray): target data
76
77     Returns:
78         float: predict percentage
79     """
80     return (pred.argmax(dim=1) == torch.tensor(y).to(device)).sum().item() / len(y)
81
82
83 def trainer(X: np.ndarray, y: np.ndarray) -> None:
84     """Trains the model with the given data
85     Args:

```