Příloha 14

Kódové provedení mlearning

LNU – třída navrženého lineárního prediktivního modelu s gradientním učením
untouched_inputs – funkce zachovává nepozměněné vstupy (pouze pro normalizaci algoritmu)
delayed_inputs – funkce pro vytvoření dalších příznaků zpožděním vstupů do LNU
delayed_inout – vytvoření dalších příznaků přidáním a zpožděním předchozích výstupů

```
# Package that contains learning algorithms for the predictors such as LNU,
LM or clasificators
# such as Logistic function for multiclas classification or binary
classification.

import numpy as np

class LNU:
    """
    Linear neuron that uses gradient descent method to estimate
    weights.

    Useful variables
    ----------
    w : array
        Numpy array of latest weights

    w_by_epoch : array
        Numpy array of weights history in each learning epoch.

    dw_by_epoch : array
        Numpy array of weights incrementation history during learning in
each epoch.

    def err_by_epoch : array
        Numpy array of error history in each epoch.

    def w_by_window : array
        Numpy array of weights history for each step in last epoch.

    def dw_by_window : array
        Numpy array of weights incrementation history for each step in last
epoch.

    def err_by_window : array
        Numpy array of error history for each step in last epoch.

    def learning_effort
        Numpy array that represents the learning activity of each data
step.
        Length of the learning_effort is corresponding to the number of
steps (i.e.
        to the length of the provided data). Each step contains absolute
value of
        weights learning incrementation (dw) that is summed up with
previous absolute
        values of dw of each learning epoch.

    """
```

```
def __init__(self, lr=.5, epochs=100, biased=True):
    self._b = (1 if biased else 0)
    self._learning_rate = lr
    self._n_epochs = epochs

    # Initialisation of future values
    self.__num_of_x = 0

    self._w = 0
    self._w_by_epoch = 0
    self._w_by_window = 0
    self._dw = 0
    self._dw_by_epoch = 0
    self._dw_by_window = 0
    self._err = 0
    self._err_by_epoch = 0
    self._err_by_window = 0

    self._learning_effort = 0

    self._yn = 0

def __learn(self, y_len, X, y, x, n_x):
    for step in range(y_len):
        x[self._b:n_x + 1] = X[step]
        self._yn[step] = np.dot(self._w, x)
        self._err = self._yn[step] - y[step]
        self._dw = -self._learning_rate * self._err * x
        self._w += self._dw

        self._w_by_window[step, :] = self._w
        self._dw_by_window[step, :] = self._dw
        self._err_by_window[step, :] = self._err

        self._learning_effort[step, :] += abs(self._dw)

def fit(self, X, y):
    y_len = len(y)
    n_weights = X.shape[1] + self._b
    n_x = X.shape[1] + self._b

    x = np.ones(n_x)

    self._yn = np.zeros(y_len)

    self._w = np.random.randn(n_weights) / n_weights
    self._err = np.zeros(y_len)

    self._w_by_window = np.zeros((y_len, n_weights))
    self._dw_by_window = np.zeros((y_len, n_weights))
    self._err_by_window = np.zeros((y_len, 1))

    self._w_by_epoch = np.zeros((self._n_epochs, n_weights))
    self._dw_by_epoch = np.zeros((self._n_epochs, n_weights))
    self._err_by_epoch = np.zeros((self._n_epochs, 1))

    self._learning_effort = np.zeros((y_len, n_weights))

    for epoch in range(self._n_epochs):
        self.__learn(y_len, X, y, x, n_x)
        self._w_by_epoch[epoch:] = self._w
```

```
            self._dw_by_epoch[epoch:] = self._dw
            self._err_by_epoch[epoch:] = self._err

        return self

    def update(self, X, y, n_epochs=1):
        y_len = len(y)
        n_weights = X.shape[1] + self._b
        n_x = X.shape[1] + self._b
        x = np.ones(n_x)

        self._learning_effort = np.zeros((y_len, n_weights))
        # self._dw_by_window = np.zeros((y_len, n_weights))

        for epoch in range(n_epochs):
            self.__learn(y_len, X, y, x, n_x)

    def fake_update(self, X, y, n_epochs=1):
        y_len = len(y)
        n_weights = X.shape[1] + self._b
        n_x = X.shape[1] + self._b
        x = np.ones(n_x)

        yn = np.zeros(y_len)
        err = 0
        w = self._w

        self._learning_effort = np.zeros((y_len, n_weights))

        for epoch in range(n_epochs):
            for step in range(y_len):
                x[self._b:n_x + 1] = X[step]
                yn[step] = np.dot(w, x)
                err = yn[step] - y[step]
                dw = -self._learning_rate * err * x
                w += dw

                self._w_by_window[step, :] = w
                self._dw_by_window[step, :] = dw
                self._learning_effort[step, :] += abs(dw)

    @property
    def w(self):
        return self._w

    def w_by_epoch(self):
        return self._w_by_epoch

    def dw_by_epoch(self):
        return self._dw_by_epoch

    def err_by_epoch(self):
        return self._err_by_epoch

    def w_by_window(self):
        return self._w_by_window

    def dw_by_window(self):
        return self._dw_by_window

    def err_by_window(self):
```

```
        return self._err_by_window

    def learning_effort(self):
        return self._learning_effort

def untouched_inputs(inputs, outputs, min_delay: int = 1, max_delay: int =
None):
    return inputs, outputs

def delayed_inputs(inputs, outputs, min_delay: int = 1, max_delay=None):
    if isinstance(max_delay, type(None)):
        max_delay = min_delay + 1

    new_array = inputs[max_delay:]
    new_output = outputs[max_delay:]
    for step in range(min_delay,max_delay):
        new_array = np.concatenate((new_array, inputs[max_delay-step:-
step]), axis=1)

    return new_array, new_output

def delayed_inout(inputs, outputs, min_delay: int = 1, max_delay: int =
None):
    if isinstance(max_delay, type(None)):
        max_delay = min_delay + 1

    new_inputs = np.concatenate((inputs, outputs), axis=1)
    new_array = new_inputs[max_delay:]
    new_output = outputs[max_delay:]
    for step in range(min_delay,max_delay):
        new_array = np.concatenate((new_array, new_inputs[max_delay-step:-
step]), axis=1)

    return new_array, new_output
```