Sieci neuronowe i sztuczna inteligencja – laboratorium 6

Monika Błyszcz, 236623

Zad 1

Zaimplementowano kod SGD:

```
# Linear Regression With Stochastic Gradient Descent for Wine Quality
from random import seed
from random import randrange
from csv import reader
from math import sqrt

# Load a CSV file
def load_csv(filename):
    dataset = list()
    with open(filename, 'r') as file:
        csv_reader = reader(file)
        for row in csv_reader:
            if not row:
                continue
            dataset.append(row)
    return dataset

# Convert string column to float
def str_column_to_float(dataset, column):
    for row in dataset:
        row[column] = float(row[column].strip())

# Find the min and max values for each column
def dataset_minmax(dataset):
    minmax = list()
    for i in range(len(dataset[0])):
        col_values = [row[i] for row in dataset]
        value_min = min(col_values)
        value_max = max(col_values)
        minmax.append([value_min, value_max])
    return minmax

# Rescale dataset columns to the range 0-1
def normalize_dataset(dataset, minmax):
    for row in dataset:
        for i in range(len(row)):
            row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] - minmax[i][0])

```

```
# Split a dataset into k folds
def cross_validation_split(dataset, n_folds):
    dataset_split = list()
    dataset_copy = list(dataset)
    fold_size = int(len(dataset) / n_folds)
    for i in range(n_folds):
        fold = list()
        while len(fold) < fold_size:
            index = randrange(len(dataset_copy))
            fold.append(dataset_copy.pop(index))
        dataset_split.append(fold)
    return dataset_split

# Calculate root mean squared error
def rmse_metric(actual, predicted):
    sum_error = 0.0
    for i in range(len(actual)):
        prediction_error = predicted[i] - actual[i]
        sum_error += (prediction_error ** 2)
    mean_error = sum_error / float(len(actual))
    return sqrt(mean_error)

# Evaluate an algorithm using a cross validation split
def evaluate_algorithm(dataset, algorithm, n_folds, *args
):
    folds = cross_validation_split(dataset, n_folds)
    scores = list()
    for fold in folds:
        train_set = list(folds)
        train_set.remove(fold)
        train_set = sum(train_set, [])
        test_set = list()
        for row in fold:
            row_copy = list(row)
            test_set.append(row_copy)
            row_copy[-1] = None
        predicted = algorithm(train_set, test_set, *args)
        actual = [row[-1] for row in fold]
        rmse = rmse_metric(actual, predicted)
        scores.append(rmse)
    return scores

# Make a prediction with coefficients
def predict(row, coefficients):
    yhat = coefficients[0]
    for i in range(len(row)-1):
        yhat += coefficients[i + 1] * row[i]
    return yhat
```

```
86
87 # Estimate linear regression coefficients using
   stochastic gradient descent
88 def coefficients_sgd(train, l_rate, n_epoch):
89     coef = [0.0 for i in range(len(train[0]))]
90     for epoch in range(n_epoch):
91         for row in train:
92             yhat = predict(row, coef)
93             error = yhat - row[-1]
94             coef[0] = coef[0] - l_rate * error
95             for i in range(len(row)-1):
96                 coef[i + 1] = coef[i + 1] - l_rate *
   error * row[i]
97             # print(l_rate, n_epoch, error)
98     return coef
99
100 # Linear Regression Algorithm With Stochastic Gradient
   Descent
101 def linear_regression_sgd(train, test, l_rate, n_epoch):
102     predictions = list()
103     coef = coefficients_sgd(train, l_rate, n_epoch)
104     for row in test:
105         yhat = predict(row, coef)
106         predictions.append(yhat)
107     return(predictions)
108
109 # Linear Regression on wine quality dataset
110 seed(1)
111 # load and prepare data
112 filename = 'winequality-white.csv'
113 dataset = load_csv(filename)
114 for i in range(len(dataset[0])):
115     str_column_to_float(dataset, i)
116
117
118 # normalize
119 minmax = dataset_minmax(dataset)
120 normalize_dataset(dataset, minmax)
121 # evaluate algorithm
122 n_folds = 5
123
124 # stała uczenia
125 l_rate = 0.01
126
```

```
127 # epoki
128 n_epoch = 500
129 scores = evaluate_algorithm(dataset,
    linear_regression_sgd, n_folds, l_rate, n_epoch)
130 print('Scores: %s' % scores)
131 print('Mean RMSE: %.3f' % (sum(scores)/float(len(scores
    ))))
```

Przeprowadzono testy na różnych wartościach epok i szybkości uczenia się. Wyniki przedstawiono w Tabeli 1.

Tabela 1: Wyniki testów

Lp	Wartość epoki	Szybkość uczenia się	RMSE
1	50	0,01	0,126
2	50	0,1	0,131
3	100	0,1	0,131
4	100	0,01	0,126
5	500	0,01	0,126

Analizując Tabelę możemy zauważyć, że nie udało się zejść z wynikiem RMSE poniżej 0.126. Oznacza, że algorytmy są do siebie zbliżone i zwiększenie wartości epoki czy szybkości uczenia się, nie wpłynie na poprawę uzyskiwanych wyników. W celu uzyskania lepszych wyników RMSE prawdopodobnie należałoby zmienić algorytm na inny.