ssxzvy3ag

December 19, 2024

[ ]:
```
import numpy as np
import pandas as pd

import random
import os
import py7zr
from tqdm import tqdm
from itertools import product
import time
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.metrics import r2_score
import pickle
```

[358]:
```
plt.style.use('ggplot')
```

[359]:
```
def set_seed(seed_value=42):
    random.seed(seed_value)
    np.random.seed(seed_value)
```

[360]:
```
spectr_dir = 'spectr_data'
zip_file_path = 'spectrums.7z'
```

[361]:
```
os.makedirs(spectr_dir, exist_ok=True)

with py7zr.SevenZipFile(zip_file_path, mode='r') as archive:
    archive.extractall(path=spectr_dir)
```

[362]:
```
cu_data = pd.read_csv('Cu_conc_in_spectrum.txt',
                      sep=' ', header=None, skiprows=1)
cu_data.columns = ['label', 'FileName']
cu_data['FileName'] = cu_data['FileName'].apply(
    lambda x: str(x).split('.')[0] + '.mca')
cu_data.head(3)
```

[362]:
```
   label FileName
0   1.75   67.mca
1   1.47   74.mca
2   1.39   83.mca
```

[363]:
```
windows = pd.read_csv('elements_windows.txt', sep=' ',
                      header=None, names=['Element', 'E1', 'E2'])
windows.head()
```

[363]:
```
  Element   E1   E2
0       S   81   87
1      Ag  105  113
2      Cr  195  201
3   Fe_Ka  227  244
4   Fe_Kb  252  267
```

[364]:
```
def parse_mca(file_path):
    with open(file_path, 'r', encoding='latin-1') as f:
        lines = f.readlines()
        data_start = False
        spectrum = []
        real_time = None

        for line in lines:
            stripped_line = line.strip()
            if stripped_line.startswith("REAL_TIME - "):
                try:
                    real_time = float(stripped_line.split("-")[1].strip())
                except ValueError:
                    pass
            if data_start:
                if stripped_line == "<<END>>":
                    break
                try:
                    spectrum.append(int(stripped_line))
                except ValueError:
                    continue
            if stripped_line == "<<DATA>>":
                data_start = True

        return np.array(spectrum), real_time
```

[365]:
```
spectra = {}
times = {}
for filename in cu_data['FileName']:
    filepath = os.path.join(spectr_dir, filename)
    if os.path.exists(filepath):
```

```
        spec, real_time = parse_mca(filepath)
        if spec is not None and real_time is not None:
            spectra[filename] = spec
            times[filename] = real_time
```

[366]:
```
df = cu_data.copy()
df['spectrum_arr'] = df['FileName'].map(spectra)
df['time'] = df['FileName'].map(times)

df = df.dropna(subset=['spectrum_arr', 'time'])

for index, row in windows.iterrows():
    element = row['Element']
    ch1 = row['E1']
    ch2 = row['E2']
    df[element] = df.apply(
        lambda row: np.sum(row['spectrum_arr'][ch1:ch2+1]) / row['time']
        if ch1 < len(row['spectrum_arr']) and ch2 < len(row['spectrum_arr'])␣
 ↪else 0,
        axis=1
    )
```

[367]:
```
df.head(3)
```

[367]:

	label	FileName	spectrum_arr	time	\
0	1.75	67.mca	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 119, 159, 135, …	59.814	
1	1.47	74.mca	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 80, 116, 103, 1…	59.744	
2	1.39	83.mca	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 78, 102, 105, 1…	60.060	

	S	Ag	Cr	Fe_Ka	Fe_Kb	Ar_Kb	…	\
0	9.429231	10.348748	8.476276	2928.177350	578.476611	4.982111	…	
1	5.908543	7.766470	9.674612	4017.959963	781.852571	3.464783	…	
2	6.310356	8.341658	9.840160	3949.067599	773.326673	3.463203	…	

	Ni_kb	Cu_Ka	Cu_Kb	Zn_Ka	Zn_Kb	Pb_La	\
0	16.718494	297.472164	48.834721	1022.369345	198.231183	96.616177	
1	16.905463	297.134440	47.017274	1272.830745	239.388056	102.822041	
2	16.650017	261.938062	44.422244	1239.110889	234.398934	100.849151	

	Pb_Lb	Ti	Nkr	Kr
0	110.509245	14.043535	1192.714080	266.442639
1	104.730182	17.323915	649.320434	176.318961
2	99.550450	17.415917	646.903097	177.439227

[3 rows x 22 columns]

$x_{norm} = \frac{x - min(x)}{max(x) - min(x)}$

[368]:
```
feature_columns = df.columns.difference(
    ['label', 'FileName', 'spectrum_arr', 'time'])

scaler = MinMaxScaler()
df[feature_columns] = scaler.fit_transform(df[feature_columns])
df['label'] = scaler.fit_transform(df[['label']])
```

[369]:
```
df = df.drop(['FileName', 'spectrum_arr', 'time'], axis=1)
df.head(3)
```

[369]:

	label	S	Ag	Cr	Fe_Ka	Fe_Kb	Ar_Kb	\
0	0.634021	0.231535	0.313065	0.545376	0.721069	0.729788	0.285071	
1	0.489691	0.012676	0.116311	0.793318	0.991717	0.989997	0.065601	
2	0.448454	0.037654	0.160137	0.827570	0.974607	0.979088	0.065373	

	Ca_Ka	Ni_Ka	Ni_kb	Cu_Ka	Cu_Kb	Zn_Ka	Zn_Kb	\
0	0.492815	0.303459	0.500176	0.187549	0.201640	0.737144	0.737183	
1	0.511980	0.064954	0.516423	0.187322	0.193414	0.920373	0.906333	
2	0.563065	0.118139	0.494226	0.163690	0.181668	0.895705	0.885828	

	Pb_La	Pb_Lb	Ti	Nkr	Kr
0	0.737475	0.556190	0.685554	0.193391	0.202682
1	0.812522	0.473542	0.929378	0.019910	0.027779
2	0.788664	0.399465	0.936216	0.019138	0.029954

[370]:
```
df.describe()
```

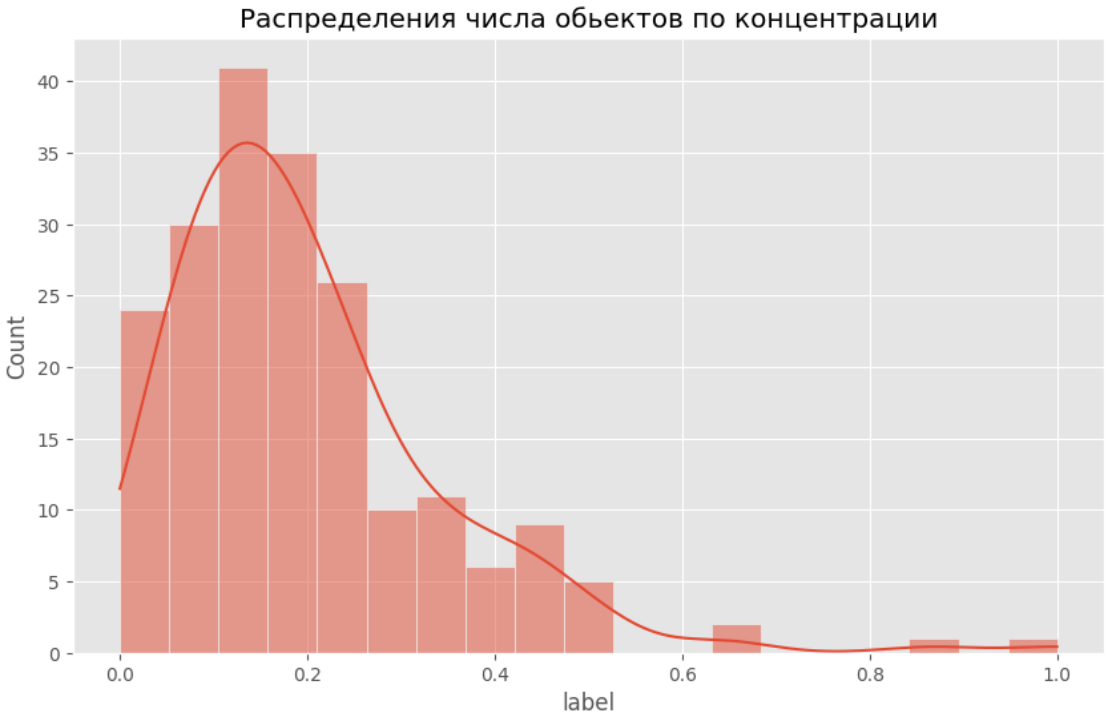
[370]:

	label	S	Ag	Cr	Fe_Ka	Fe_Kb	\
count	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	
mean	0.200595	0.230291	0.284822	0.375099	0.630514	0.640909	
std	0.146930	0.151800	0.136046	0.174398	0.172235	0.164677	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.097938	0.112705	0.174042	0.246380	0.503216	0.523329	
50%	0.164948	0.208903	0.271308	0.372684	0.621928	0.640001	
75%	0.252577	0.331261	0.371167	0.495683	0.775934	0.782149	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

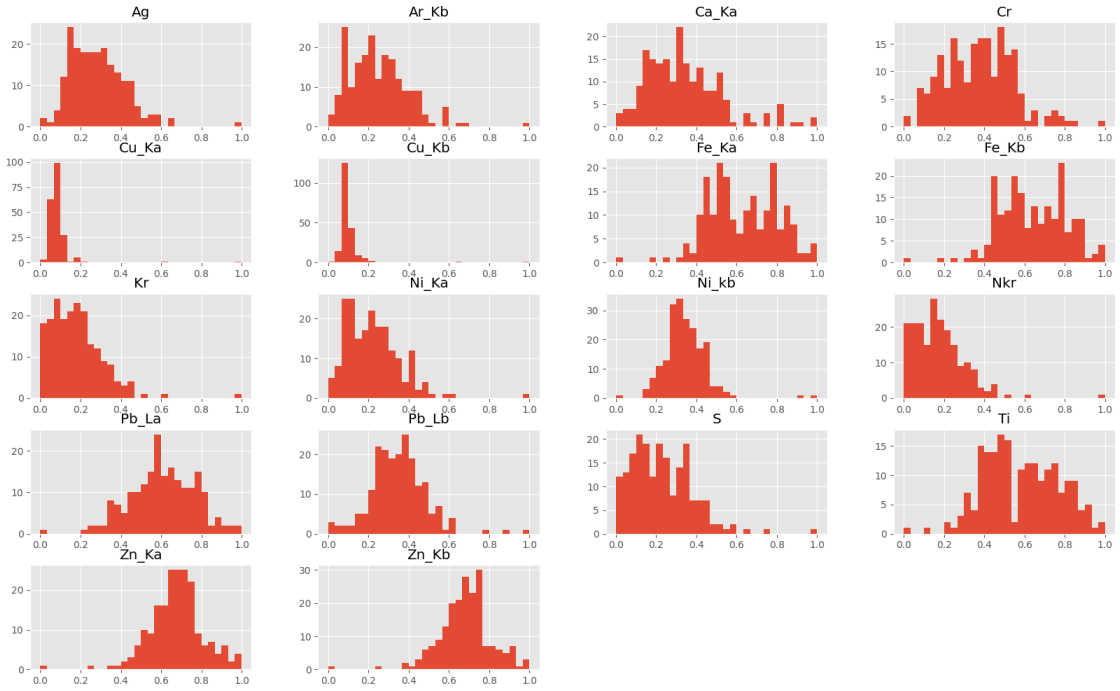
	Ar_Kb	Ca_Ka	Ni_Ka	Ni_kb	Cu_Ka	Cu_Kb	\
count	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	
mean	0.251738	0.338669	0.222193	0.344789	0.086774	0.103093	
std	0.146517	0.191429	0.133550	0.106521	0.080638	0.079354	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.149047	0.195586	0.115613	0.286043	0.064210	0.080925	
50%	0.231834	0.317011	0.206099	0.332797	0.074901	0.090182	
75%	0.333859	0.435699	0.295079	0.399016	0.086319	0.105747	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

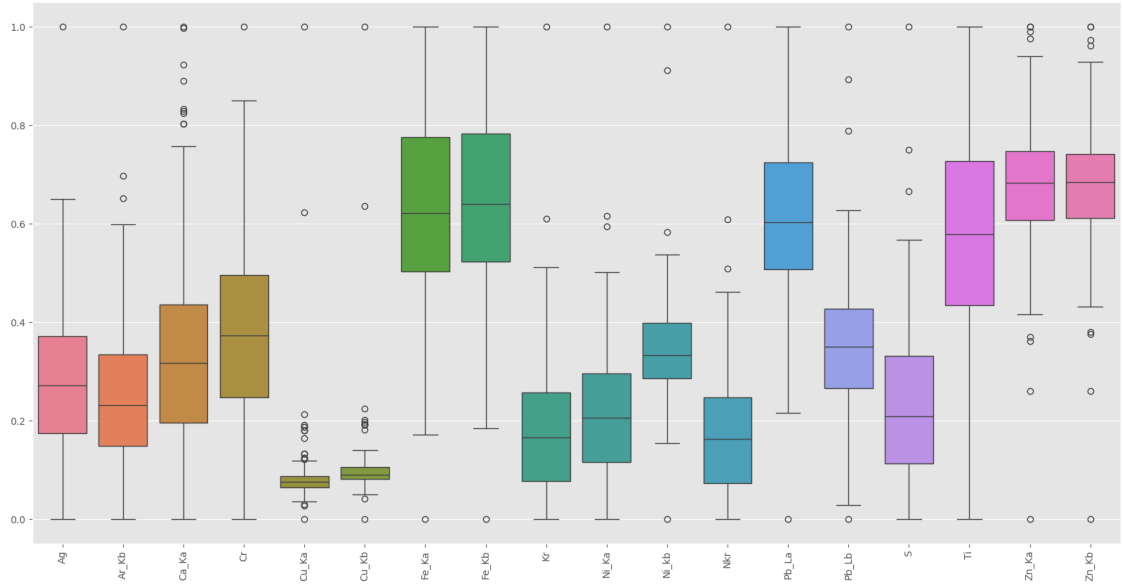
```
            Zn_Ka       Zn_Kb       Pb_La       Pb_Lb          Ti          Nkr  \
count  201.000000  201.000000  201.000000  201.000000  201.000000  201.000000
mean     0.676519    0.680647    0.606594    0.351848    0.581155    0.176563
std      0.132710    0.129243    0.162604    0.136531    0.183893    0.128392
min      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
25%      0.606852    0.611396    0.507341    0.265187    0.434641    0.073112
50%      0.682563    0.684799    0.603091    0.349250    0.577936    0.162791
75%      0.747391    0.740588    0.723667    0.427666    0.726734    0.246323
max      1.000000    1.000000    1.000000    1.000000    1.000000    1.000000

               Kr
count  201.000000
mean     0.180288
std      0.128960
min      0.000000
25%      0.076721
50%      0.166143
75%      0.256836
max      1.000000
```

[371]:
```
plt.figure(figsize=(10, 6))
sns.histplot(df['label'], kde=True)
plt.title('                          ')
plt.show()
```



[372]:
```
df[feature_columns].hist(bins=30, figsize=(20, 12))
plt.show()

plt.figure(figsize=(20, 10))
sns.boxplot(data=df[feature_columns])
plt.xticks(rotation=90)
plt.show()
```





[373]:
```
df.head(1)
```

[373]:

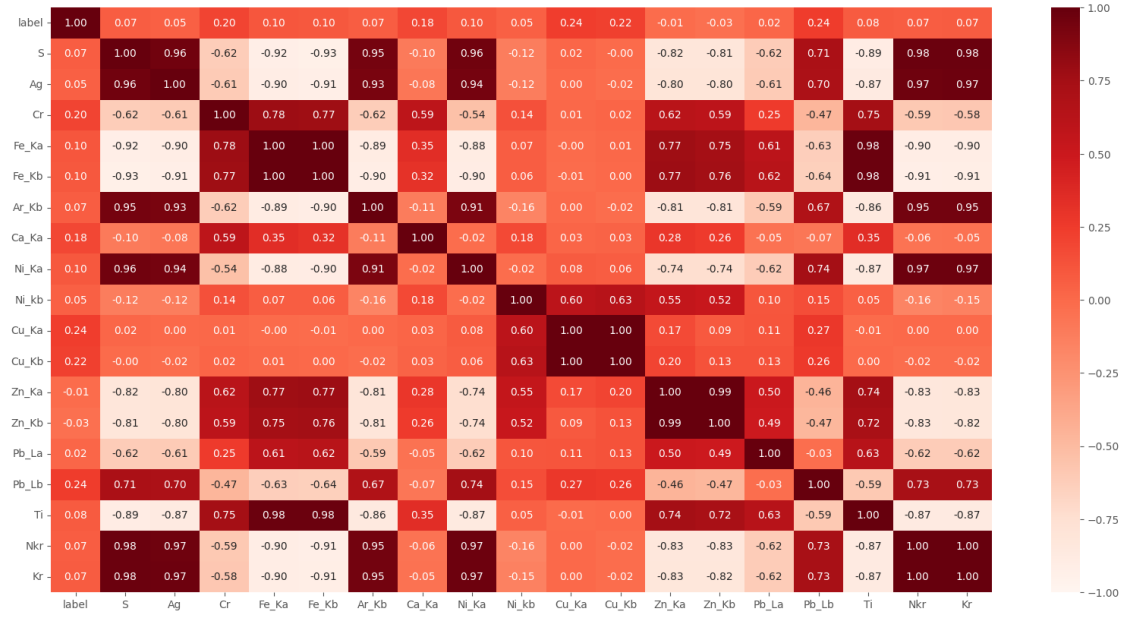
	label	S	Ag	Cr	Fe_Ka	Fe_Kb	Ar_Kb	\
0	0.634021	0.231535	0.313065	0.545376	0.721069	0.729788	0.285071	

	Ca_Ka	Ni_Ka	Ni_kb	Cu_Ka	Cu_Kb	Zn_Ka	Zn_Kb	\
0	0.492815	0.303459	0.500176	0.187549	0.20164	0.737144	0.737183	

	Pb_La	Pb_Lb	Ti	Nkr	Kr
0	0.737475	0.55619	0.685554	0.193391	0.202682

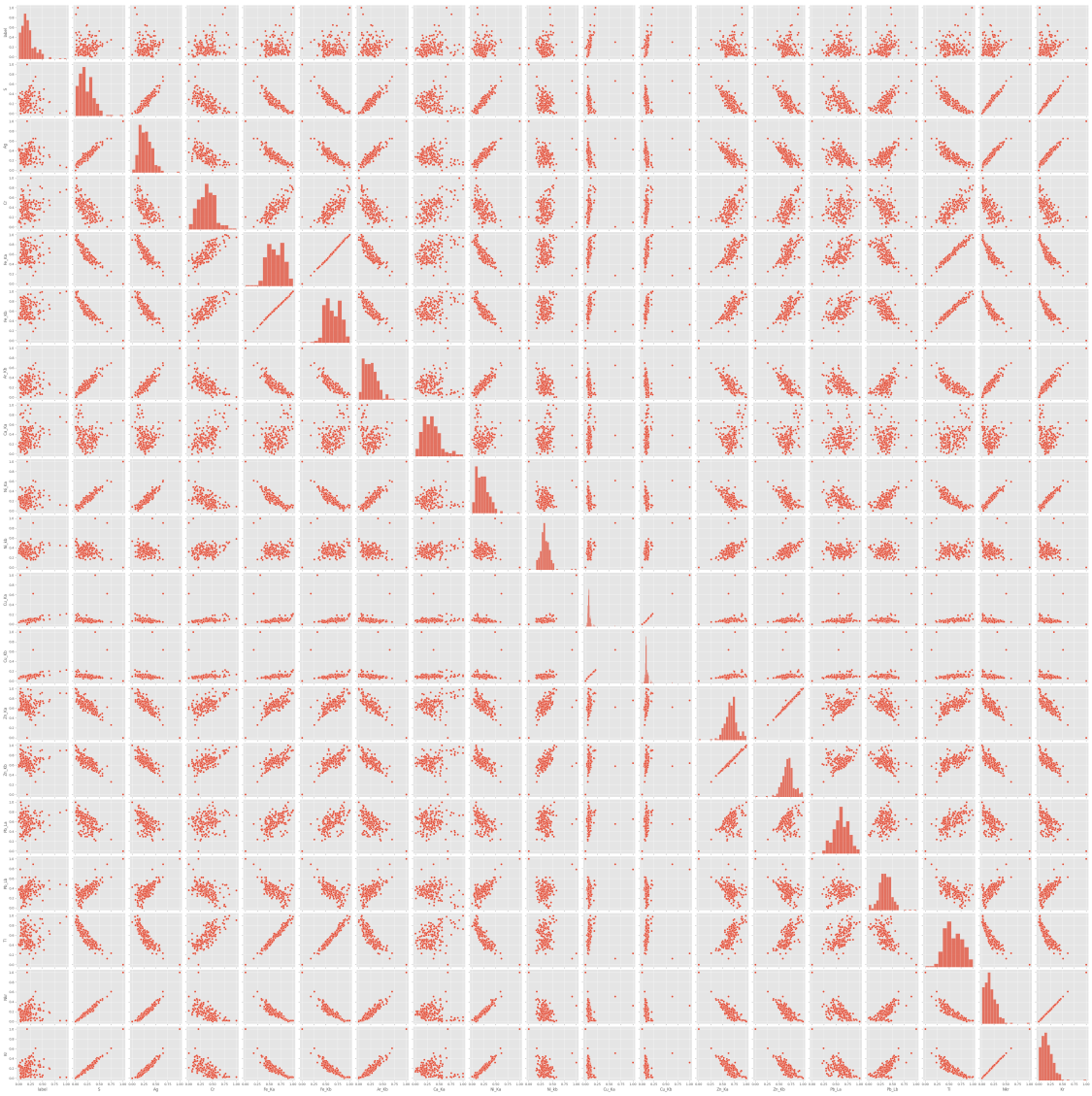
[374]:
```
n_cols = 6
n_rows = (len(feature_columns) + n_cols - 1) // n_cols

fig, axes = plt.subplots(n_rows, n_cols, figsize=(n_cols * 6, n_rows * 6))

axes = axes.flatten()

for i, feature in enumerate(feature_columns):
    sns.scatterplot(x=df[feature], y=df['label'], ax=axes[i])
    axes[i].set_title(f'{feature} vs Concentration')
    axes[i].set_xlabel(feature)
    axes[i].set_ylabel('Concentration')

for i in range(len(feature_columns), len(axes)):
    fig.delaxes(axes[i])
plt.tight_layout()
plt.show()
```

[375]:
```
correlation_matrix = df.corr()

plt.figure(figsize=(20, 10))
sns.heatmap(correlation_matrix, annot=True,
            cmap='Reds', fmt='.2f', vmin=-1, vmax=1)
plt.show()
```



[376]:
```
sns.pairplot(df)
```

[376]:
```
<seaborn.axisgrid.PairGrid at 0x238b3148f20>
```



[380]:
```
df.to_csv('dataset.csv')
```

[ ]:
```
X = df[feature_columns]
y = df['label']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=4)
```

[382]:
```
training_data = [(np.array(x).reshape(-1, 1), y)
                 for x, y in zip(X_train.values.tolist(), y_train.values.
 ↪tolist())]
```

```
test_data = [(np.array(x).reshape(-1, 1), y)
             for x, y in zip(X_test.values.tolist(), y_test.values.tolist())]
```

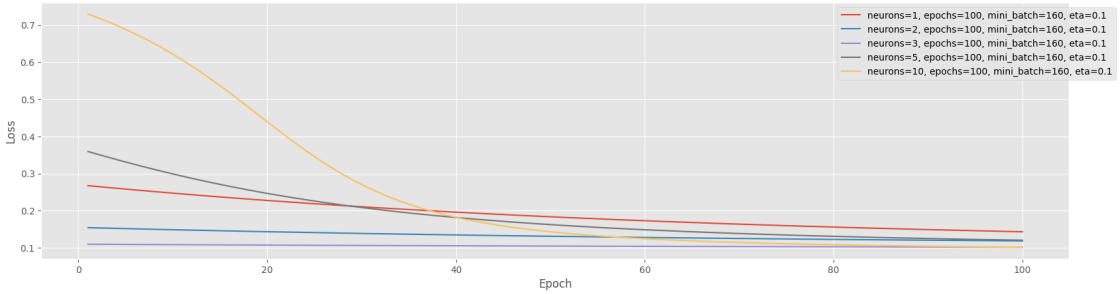
[451]:
```
class Network(object):
    def __init__(self, sizes):
        set_seed(42)
        self.num_layers = len(sizes)
        self.sizes = sizes
        self.biases = [np.random.randn(y, 1) for y in sizes[1:]]
        self.weights = [np.random.randn(y, x)
                        for x, y in zip(sizes[:-1], sizes[1:])]

    def feedforward(self, a):
        for b, w in zip(self.biases, self.weights):
            a = self.sigmoid(np.dot(w, a) + b)
        return a

    def SGD(self, training_data, epochs, mini_batch_size, eta, test_data=None):
        set_seed(42)
        n = len(training_data)
        epoch_train_losses = []  #
        epoch_test_rmse = []     #       (        RMSE)

        with tqdm.tqdm(range(epochs), desc="Training Progress") as pbar:
            for j in pbar:
                random.shuffle(training_data)
                mini_batches = [training_data[k:k + mini_batch_size]
                                for k in range(0, n, mini_batch_size)]
                for mini_batch in mini_batches:
                    self.update_mini_batch(mini_batch, eta)

                epoch_train_loss = self.calculate_loss(training_data)
                epoch_train_losses.append(epoch_train_loss)

                #                ,       RMSE
                if test_data:
                    epoch_rmse = self.calculate_rmse(test_data)
                    epoch_test_rmse.append(epoch_rmse)

                #             tqdm
                if test_data:
                    pbar.set_postfix(
                        train_loss=epoch_train_loss, test_rmse=epoch_rmse)
                else:
                    pbar.set_postfix(train_loss=epoch_train_loss)

        #                    RMSE
```

```
        return epoch_train_losses, epoch_test_rmse

    def update_mini_batch(self, mini_batch, eta):
        nabla_b = [np.zeros(b.shape) for b in self.biases]
        nabla_w = [np.zeros(w.shape) for w in self.weights]
        for x, y in mini_batch:
            delta_nabla_b, delta_nabla_w = self.backprop(x, y)
            nabla_b = [nb + dnb for nb, dnb in zip(nabla_b, delta_nabla_b)]
            nabla_w = [nw + dnw for nw, dnw in zip(nabla_w, delta_nabla_w)]
        self.weights = [w - (eta / len(mini_batch)) *
                        nw for w, nw in zip(self.weights, nabla_w)]
        self.biases = [b - (eta / len(mini_batch)) * nb for b,
                       nb in zip(self.biases, nabla_b)]

    def backprop(self, x, y):
        nabla_b = [np.zeros(b.shape) for b in self.biases]
        nabla_w = [np.zeros(w.shape) for w in self.weights]

        activation = x
        activations = [x]
        zs = []
        for b, w in zip(self.biases, self.weights):
            z = np.dot(w, activation) + b
            zs.append(z)
            activation = self.sigmoid(z)
            activations.append(activation)

        delta = self.cost_derivative(
            activations[-1], y) * self.sigmoid_prime(zs[-1])
        nabla_b[-1] = delta
        nabla_w[-1] = np.dot(delta, activations[-2].transpose())

        for l in range(2, self.num_layers):
            z = zs[-l]
            sp = self.sigmoid_prime(z)
            delta = np.dot(self.weights[-l + 1].transpose(), delta) * sp
            nabla_b[-l] = delta
            nabla_w[-l] = np.dot(delta, activations[-l - 1].transpose())
        return (nabla_b, nabla_w)

    def evaluate(self, test_data):
        test_results = [(np.argmax(self.feedforward(x)), y)
                        for (x, y) in test_data]
        return sum(int(x == y) for (x, y) in test_results)

    def cost_derivative(self, output_activations, y):
        return (output_activations - y)
```

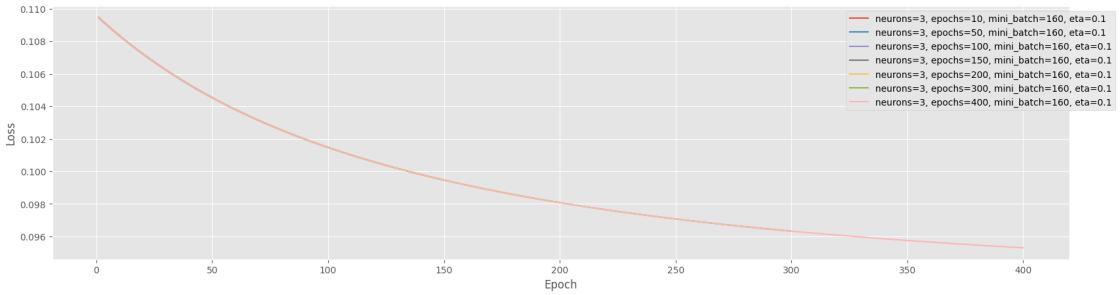
```
    def sigmoid(self, z):
        return 1.0 / (1.0 + np.exp(-z))

    def sigmoid_prime(self, z):
        return self.sigmoid(z) * (1 - self.sigmoid(z))

    def calculate_loss(self, data):
        loss = 0
        for x, y in data:
            output = self.feedforward(x)
            loss += np.sum((output - y) ** 2)
        return loss / len(data)

    def calculate_rmse(self, test_data):
        actual = []
        predicted = []
        for x, y in test_data:
            actual.append(y)
            predicted.append(self.feedforward(x))
        actual = np.array(actual).flatten()
        predicted = np.array(predicted).flatten()
        rmse = np.sqrt(mean_squared_error(actual, predicted))
        return rmse
```

[416]:
```
def train_and_plot_loss(training_data, test_data, feature_columns,␣
↪neuron_counts, epochs_list, mini_batch_sizes, eta_list):
    plt.figure(figsize=(20, 5))

    for neurons in neuron_counts:
        for epochs in epochs_list:
            for mini_batch_size in mini_batch_sizes:
                for eta in eta_list:
                    net = Network([len(feature_columns), neurons, 1])
                    epoch_losses = net.SGD(
                        training_data, epochs=epochs,␣
↪mini_batch_size=mini_batch_size, eta=eta, test_data=test_data)

                    label = f"neurons={neurons}, epochs={
                        epochs}, mini_batch={mini_batch_size}, eta={eta}"
                    plt.plot(range(1, len(epoch_losses) + 1),
                             epoch_losses, label=label)
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend(loc='upper right', bbox_to_anchor=(1.05, 1))
    plt.grid(True)
    plt.show()
```

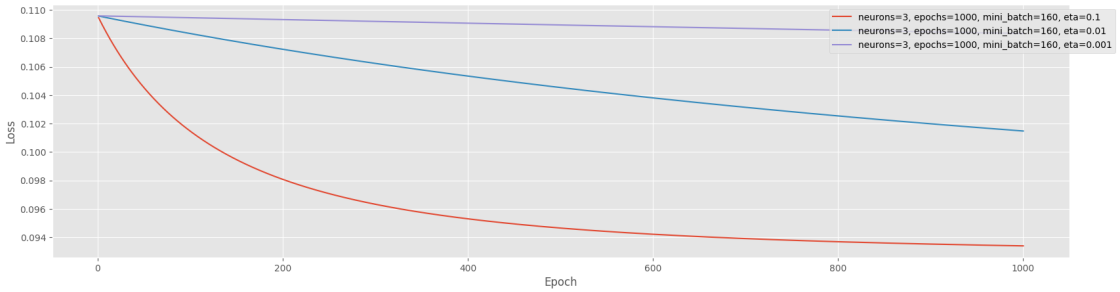
[252]:
```
neurons_list = [1, 2, 3, 5, 10]
epochs_list = [100]
mini_batch_sizes = [len(training_data)]
eta_list = [0.1]
train_and_plot_loss(training_data, test_data, feature_columns,
                    neurons_list, epochs_list, mini_batch_sizes, eta_list)
```



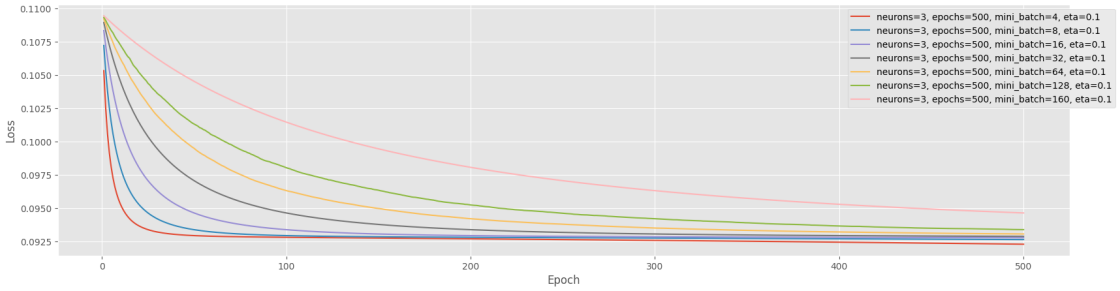
[253]:
```
neurons_list = [3]
epochs_list = [10, 50, 100, 150, 200, 300, 400]
mini_batch_sizes = [len(training_data)]
eta_list = [0.1]
train_and_plot_loss(training_data, test_data, feature_columns,
                    neurons_list, epochs_list, mini_batch_sizes, eta_list)
```



[251]:
```
neurons_list = [3]
epochs_list = [1000]
mini_batch_sizes = [len(training_data)]
eta_list = [0.1, 0.01, 0.001]
train_and_plot_loss(training_data, test_data, feature_columns,
                    neurons_list, epochs_list, mini_batch_sizes, eta_list)
```



[254]:
```
neurons_list = [3]
epochs_list = [500]
mini_batch_sizes = [4, 8, 16, 32, 64, 128, len(training_data)]
eta_list = [0.1]
train_and_plot_loss(training_data, test_data, feature_columns,
                    neurons_list, epochs_list, mini_batch_sizes, eta_list)
```



$$R_2 = 1 - \frac{\sum_i (y_i - y_{pred,i}^2)}{\sum_i (y_i - \mu)^2}$$

[385]:
```
net = Network([len(feature_columns), 16, 8, 1])
epoch_losses = net.SGD(training_data, epochs=300,
                       mini_batch_size=4, eta=0.01, test_data=test_data)
```

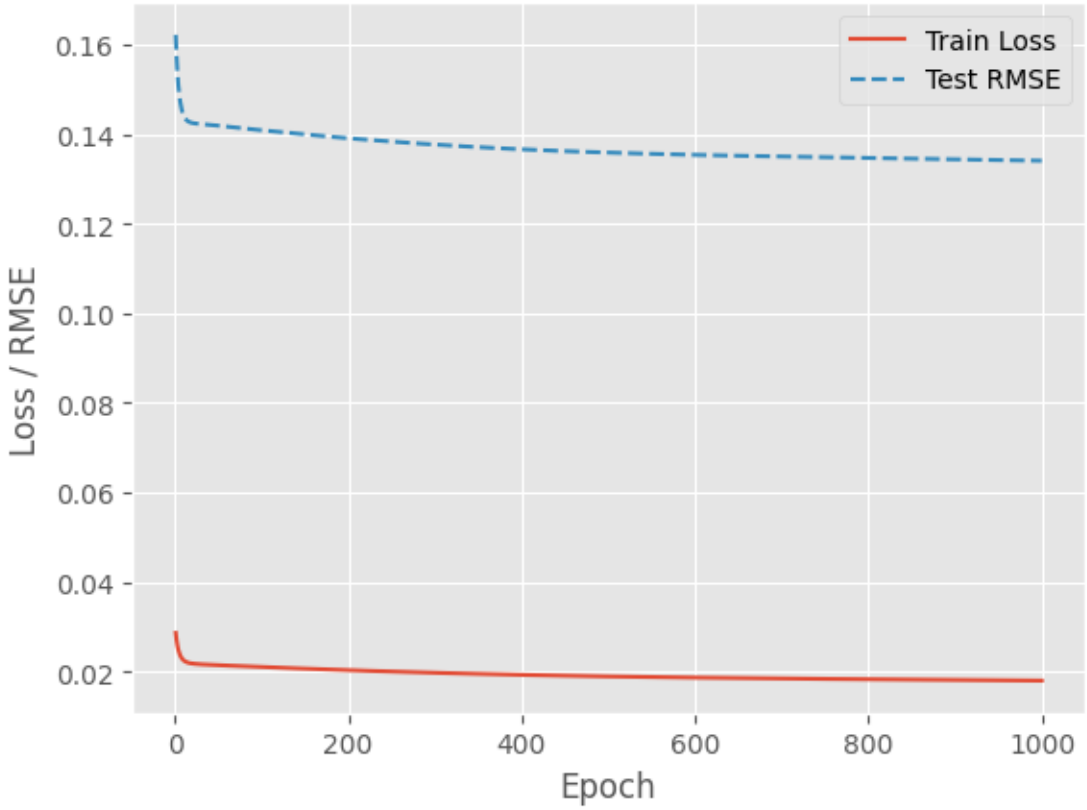
```
Training Progress: 100%|      | 300/300 [00:08<00:00, 34.74epoch/s,
Loss=0.0216]
```

[452]:
```
def plot_rmse(train_losses, test_rmse=None):
    plt.plot(range(1, len(train_losses) + 1), train_losses, label='Train Loss')
    if test_rmse:
        plt.plot(range(1, len(test_rmse) + 1), test_rmse,
                 label='Test RMSE', linestyle='dashed')
    plt.xlabel('Epoch')
    plt.ylabel('Loss / RMSE')
    plt.grid(True)
    plt.legend(loc='best')
```

```
    plt.show()
```

[471]:
```
net = Network([len(feature_columns), 50, 25, 1])

train_losses, test_rmse = net.SGD(
    training_data, epochs=1000, mini_batch_size=8, eta=0.01,␣
 ↪test_data=test_data)
plot_rmse(train_losses, test_rmse)
```

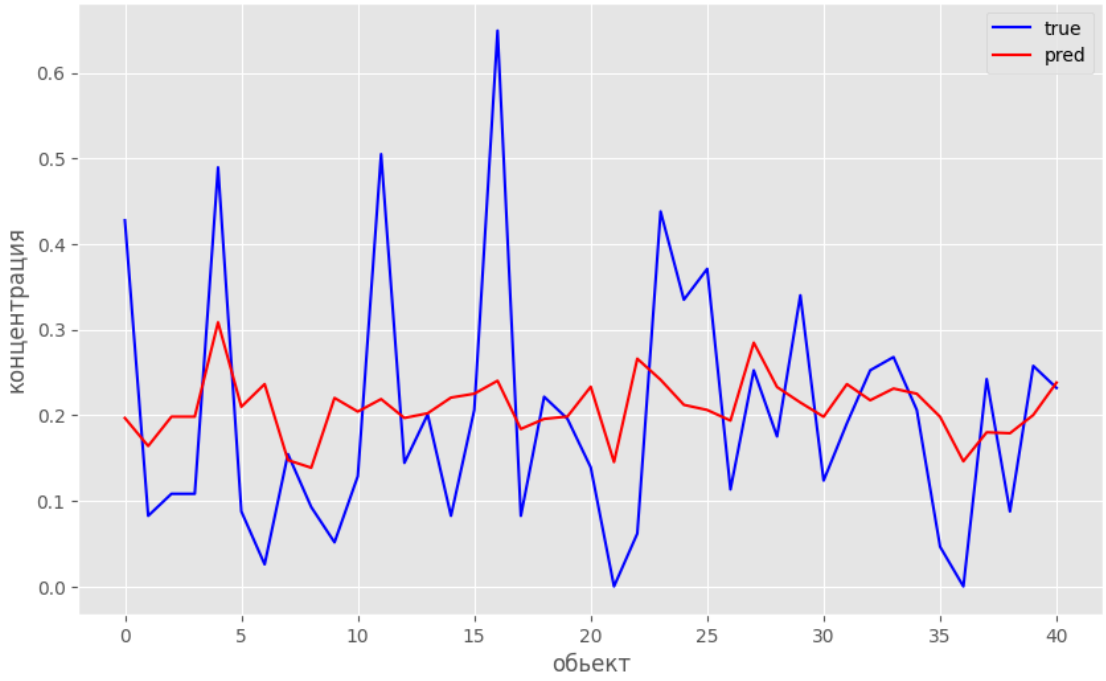
```
Training Progress: 100%|      | 1000/1000 [00:30<00:00, 32.26it/s,
test_rmse=0.134, train_loss=0.0181]
```



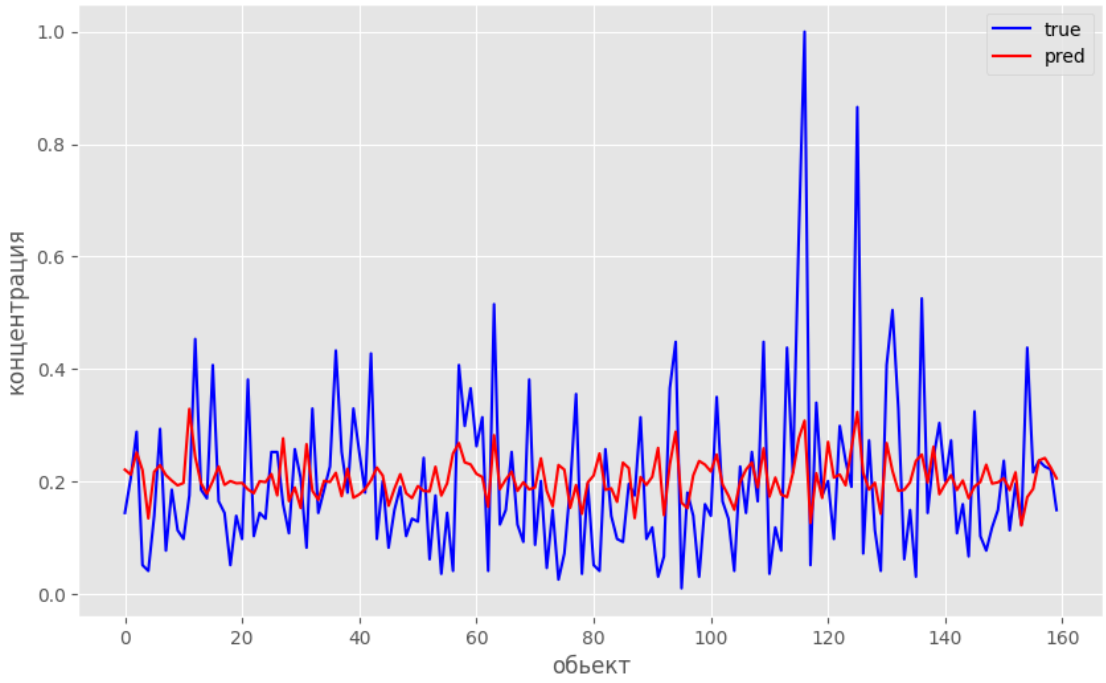
[472]:
```
def get_predictions(model, test_data):
    predictions = []
    for x, _ in test_data:
        predictions.append(model.feedforward(x))
    return predictions


def plot_predictions(test_data, predictions):
    true_values = [y for _, y in test_data]
```

```
    predicted_values = [pred[0] for pred in predictions]

    plt.figure(figsize=(10, 6))
    plt.plot(range(len(test_data)), true_values,
             label='true', color='blue')
    plt.plot(range(len(test_data)), predicted_values,
             label='pred', color='red')
    plt.xlabel('    ')
    plt.ylabel('       ')
    plt.legend()
    plt.grid(True)
    plt.show()
```

[473]:
```
predictions = get_predictions(net, test_data)
plot_predictions(test_data, predictions)
```



[474]:
```
predictions = get_predictions(net, training_data)
plot_predictions(training_data, predictions)
```



[475]:
```
from sklearn.metrics import mean_squared_error
test_results = [(net.feedforward(x), y) for (x, y) in test_data]
predicted = [result[0][0] for result in test_results]
actual = [result[1] for result in test_results]

rmse = np.sqrt(mean_squared_error(actual, predicted))
print(f"Root Mean Squared Error: {rmse}")
```

```
Root Mean Squared Error: 0.1340757281120462
```

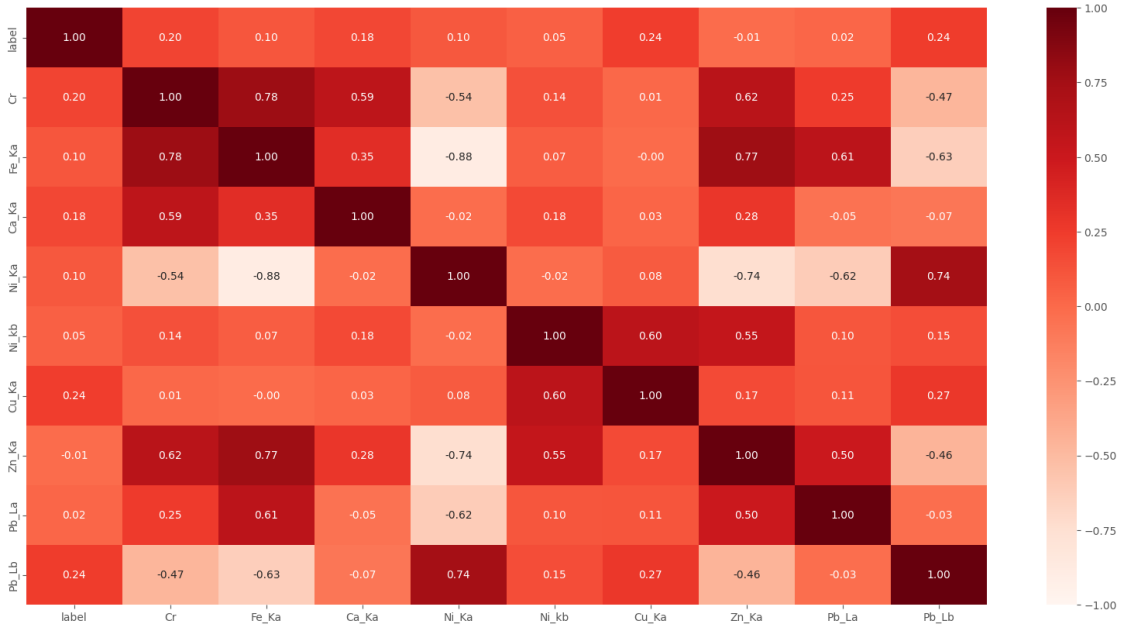
[506]:
```
correlation_matrix = df.drop(columns=['label']).corr()
label_correlation = df.corr()['label']
columns_to_drop = set()
for col1 in correlation_matrix.columns:
    for col2 in correlation_matrix.columns:
        if col1 != col2 and (correlation_matrix.loc[col1, col2] > 0.8):
            if label_correlation[col1] >= label_correlation[col2]:
                columns_to_drop.add(col2)
            else:
                columns_to_drop.add(col1)

df1 = df.drop(columns=columns_to_drop)

df1.head()
```

[506]:

	label	Cr	Fe_Ka	Ca_Ka	Ni_Ka	Ni_kb	Cu_Ka	\
0	0.634021	0.545376	0.721069	0.492815	0.303459	0.500176	0.187549	
1	0.489691	0.793318	0.991717	0.511980	0.064954	0.516423	0.187322	
2	0.448454	0.827570	0.974607	0.563065	0.118139	0.494226	0.163690	
3	1.000000	0.764076	1.000000	0.652656	0.108286	0.443447	0.212785	
4	0.865979	0.712293	0.966873	0.756323	0.121018	0.450786	0.191769	

	Zn_Ka	Pb_La	Pb_Lb
0	0.737144	0.737475	0.556190
1	0.920373	0.812522	0.473542
2	0.895705	0.788664	0.399465
3	0.906779	0.836439	0.464816
4	0.908752	0.789741	0.479420

[509]:
```
correlation_matrix1 = df1.corr()

plt.figure(figsize=(20, 10))
sns.heatmap(correlation_matrix1, annot=True,
            cmap='Reds', fmt='.2f', vmin=-1, vmax=1)
plt.show()
```

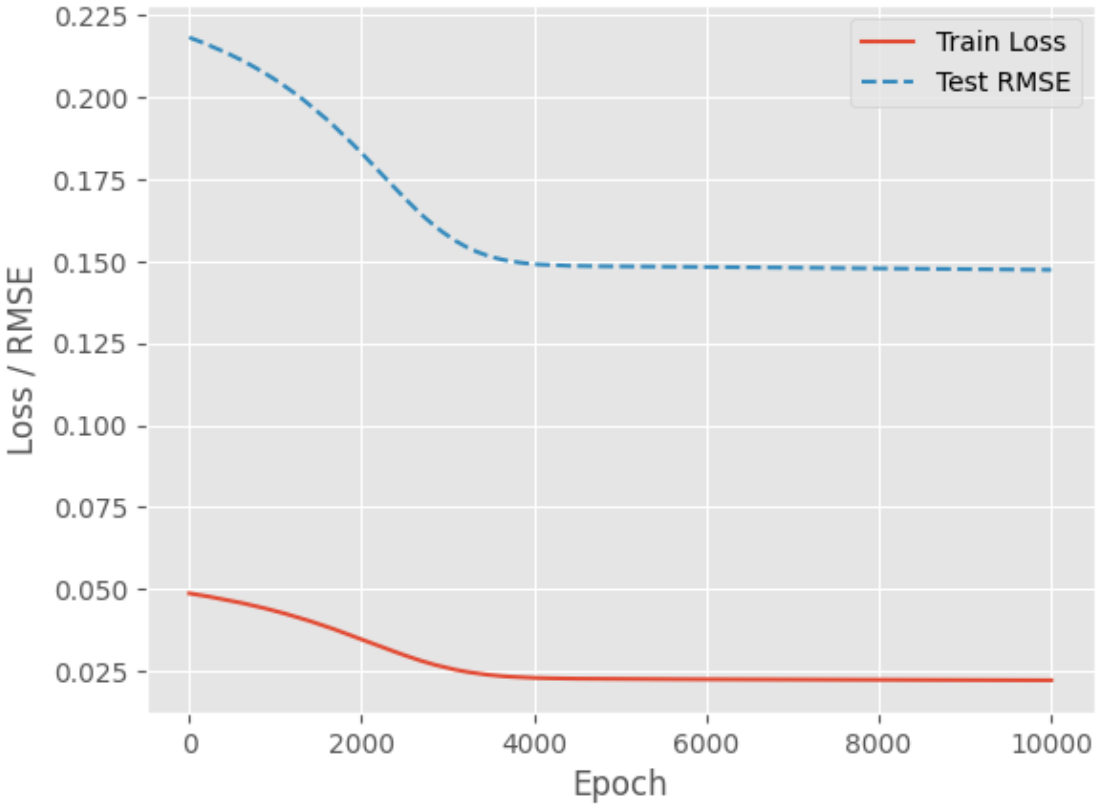


[511]:
```
X = df1.drop(columns=['label'])
y = df1['label']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=4)
```

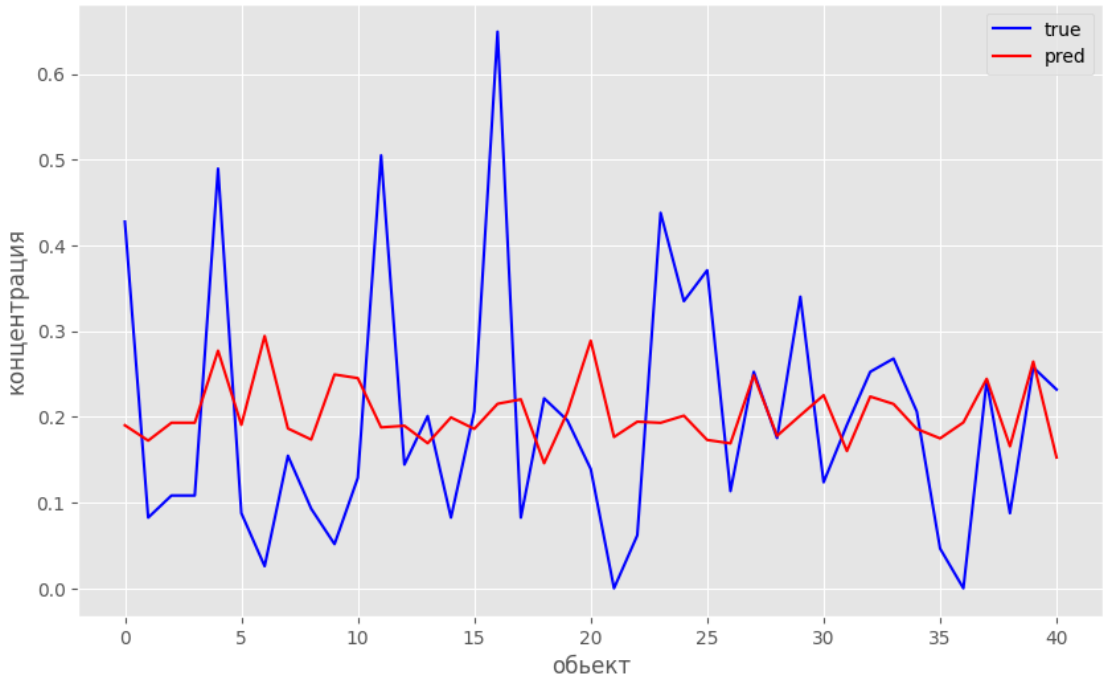
[512]:
```
training_data = [(np.array(x).reshape(-1, 1), y)
                 for x, y in zip(X_train.values.tolist(), y_train.values.
 ↪tolist())]
test_data = [(np.array(x).reshape(-1, 1), y)
             for x, y in zip(X_test.values.tolist(), y_test.values.tolist())]
```

[518]:
```
net1 = Network([X.shape[1], 50, 25, 1])

train_losses, test_rmse = net1.SGD(
    training_data, epochs=10000, mini_batch_size=64, eta=0.001,␣
 ↪test_data=test_data)
plot_rmse(train_losses, test_rmse)
```

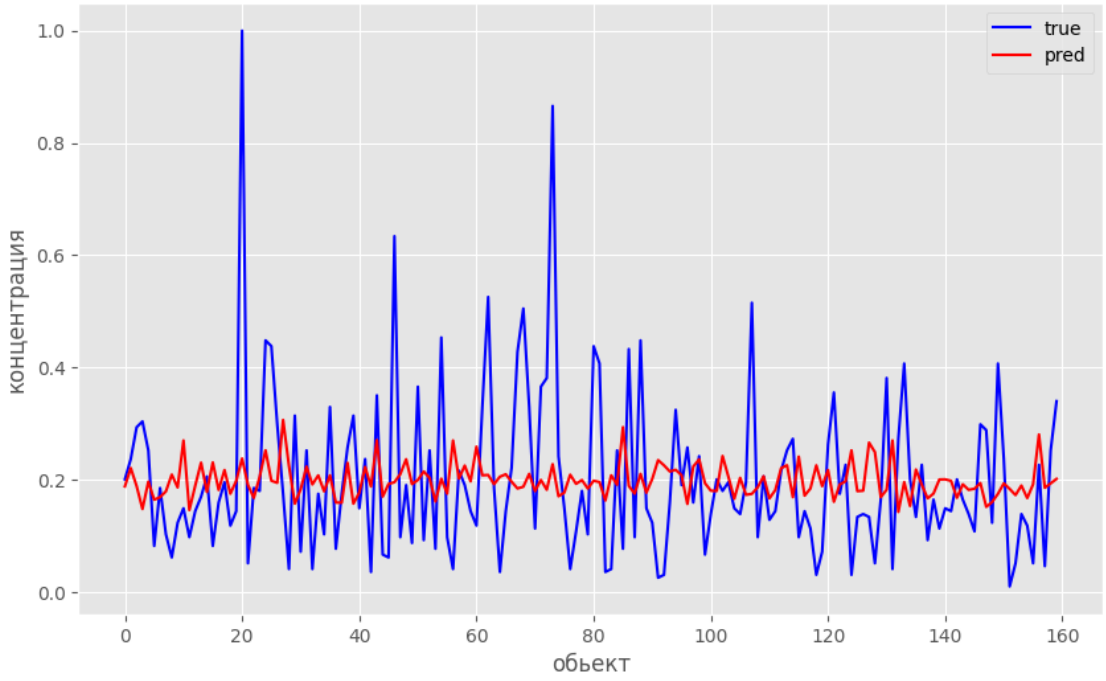
```
Training Progress: 100%|      | 10000/10000 [05:10<00:00, 32.22it/s,
test_rmse=0.147, train_loss=0.0222]
```



[519]:
```
predictions = get_predictions(net1, test_data)
plot_predictions(test_data, predictions)
```



[520]:
```
predictions = get_predictions(net1, training_data)
plot_predictions(training_data, predictions)
```



[521]:
```
test_results = [(net1.feedforward(x), y) for (x, y) in test_data]
predicted = [result[0][0] for result in test_results]
actual = [result[1] for result in test_results]

rmse = np.sqrt(mean_squared_error(actual, predicted))
print(f"Root Mean Squared Error: {rmse}")
```

```
Root Mean Squared Error: 0.14739070775808072
```