KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

Intelektikos pagrindai (P176B101)

Laboratorinių darbų ataskaita

Atliko:
IFF-1/4 gr. studentas
Mildaras Karvelis
2024 m. gegužės 8 d.
Priėmė:
lekt. Audrius Nečiūnas

KAUNAS 2023

TURINYS

1. Įvadas

Darbo metu bus panaudotas paprasčiausios struktūros dirbtinis neuroninis tinklas – vienetinis neuronas su tiesine aktyvavimo funkcija (purelin(n)=purelin(Wp+b)=Wp+b). Neurono užduotimi bus laiko eilutės k-osios reikšmės a(k) prognozavimas panaudojant n ankstesnes reikšmes a(k-1), a(k-2), ..., a(k-n). Modelį, kurį realizuojame esant prielaidai, kad priklausomybė tarp prognozuojamos reikšmės ir prieš tai esančių eilės elementų gali būti aprašyta naudojant tiesinę funkciją, vadiname autoregresiniu tiesiniu modeliu n-tosios eilės.

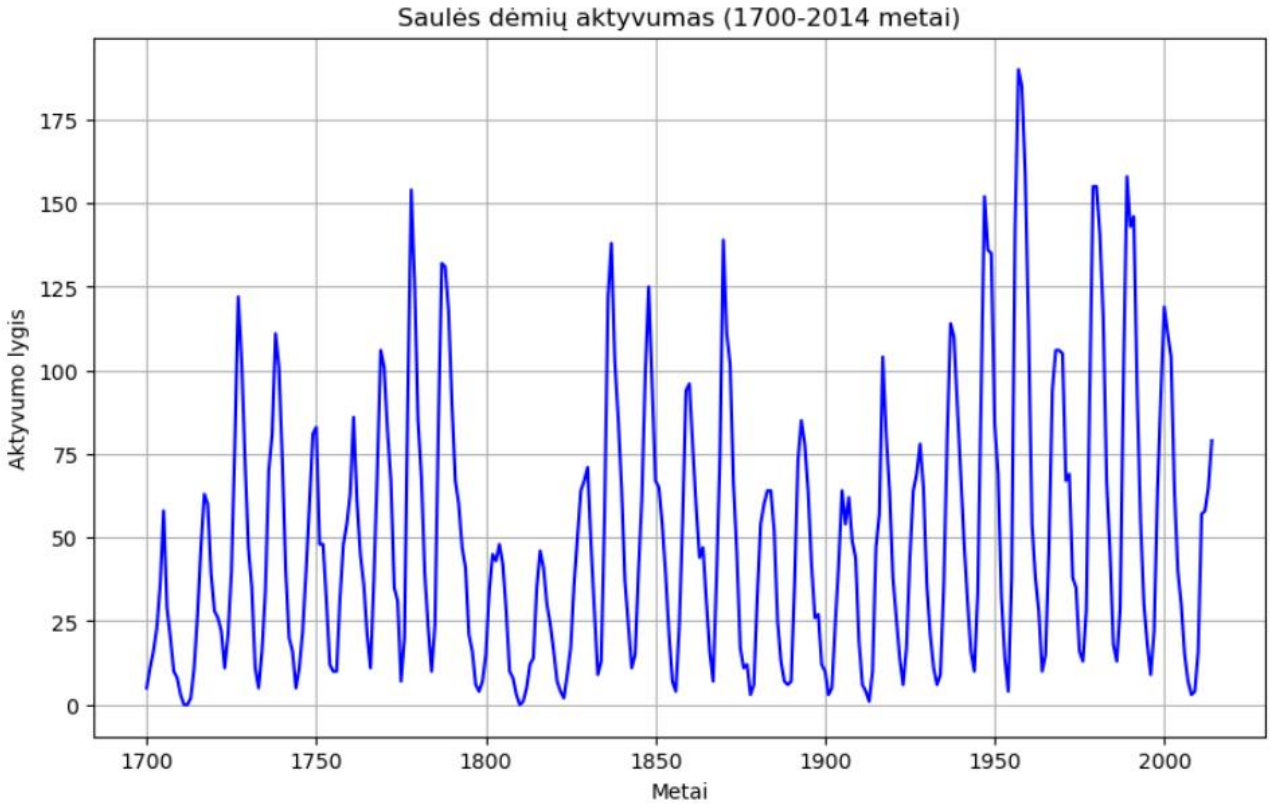
2. Duomenų rinkinys naudotas II daliai

Duomenų rinkinį sudaro:

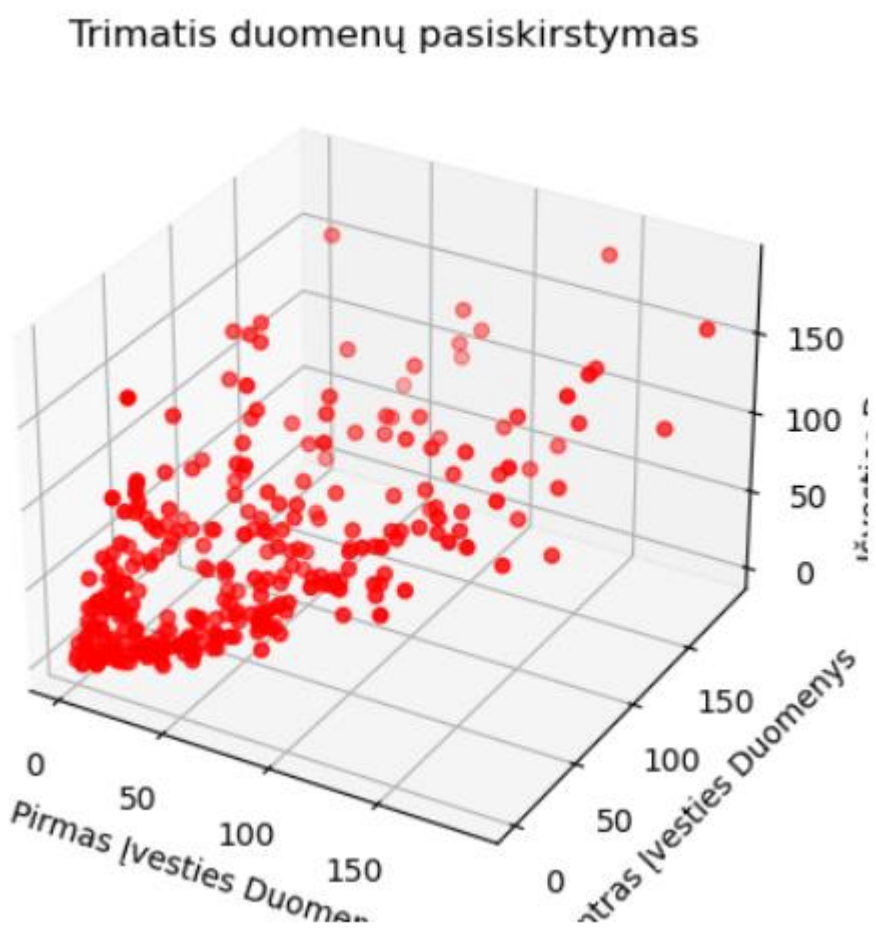
- „City“ – miesto pavadinimas;
- „Vehicle Type“ – transporto priemonės tipas;
- „Weather“ – oro sąlygos;
- „Economic Condition“ – ekonominė padėtis;
- „Day of Week“ – savaitės diena;
- „Hour of Day“ – valanda (1-24 h.);
- „Speed“ – greitis, km/h;
- „Is Peak Hour “ – ar tai piko valanda? (True arba false, 0 arba 1);
- „Random Event Occured“ – ar kažkas įvyko? (True arba false, 0 arba 1);
- „Energy Consumption“ – energijos suvartojimas;
- „Traffic Density“ – eismo tankumas.

3. 1 dalis.

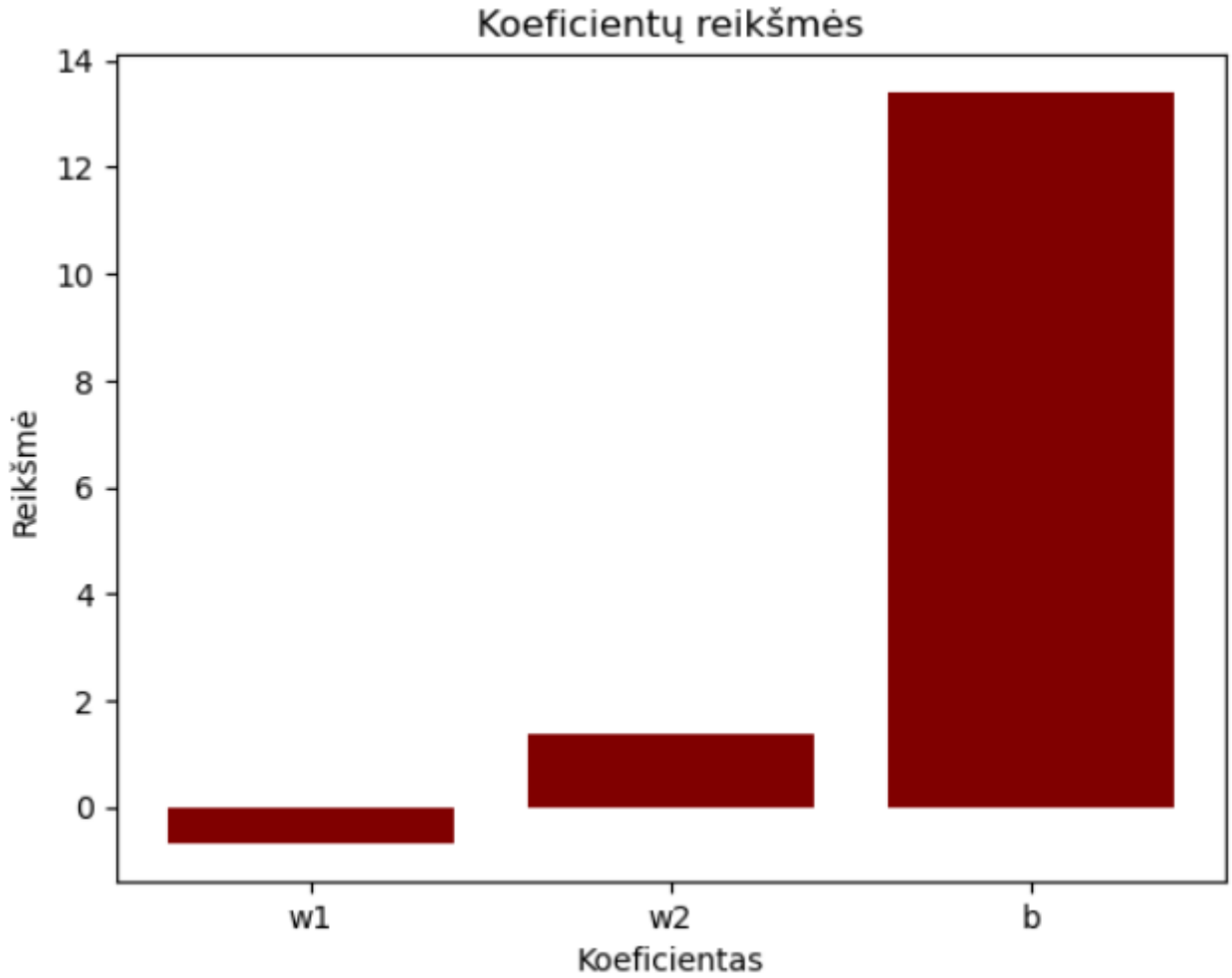
3.1. Saulės dėmių aktyvumo už 1700- 2014 metus grafikas



3.2. Trimatis duomenų grafikas

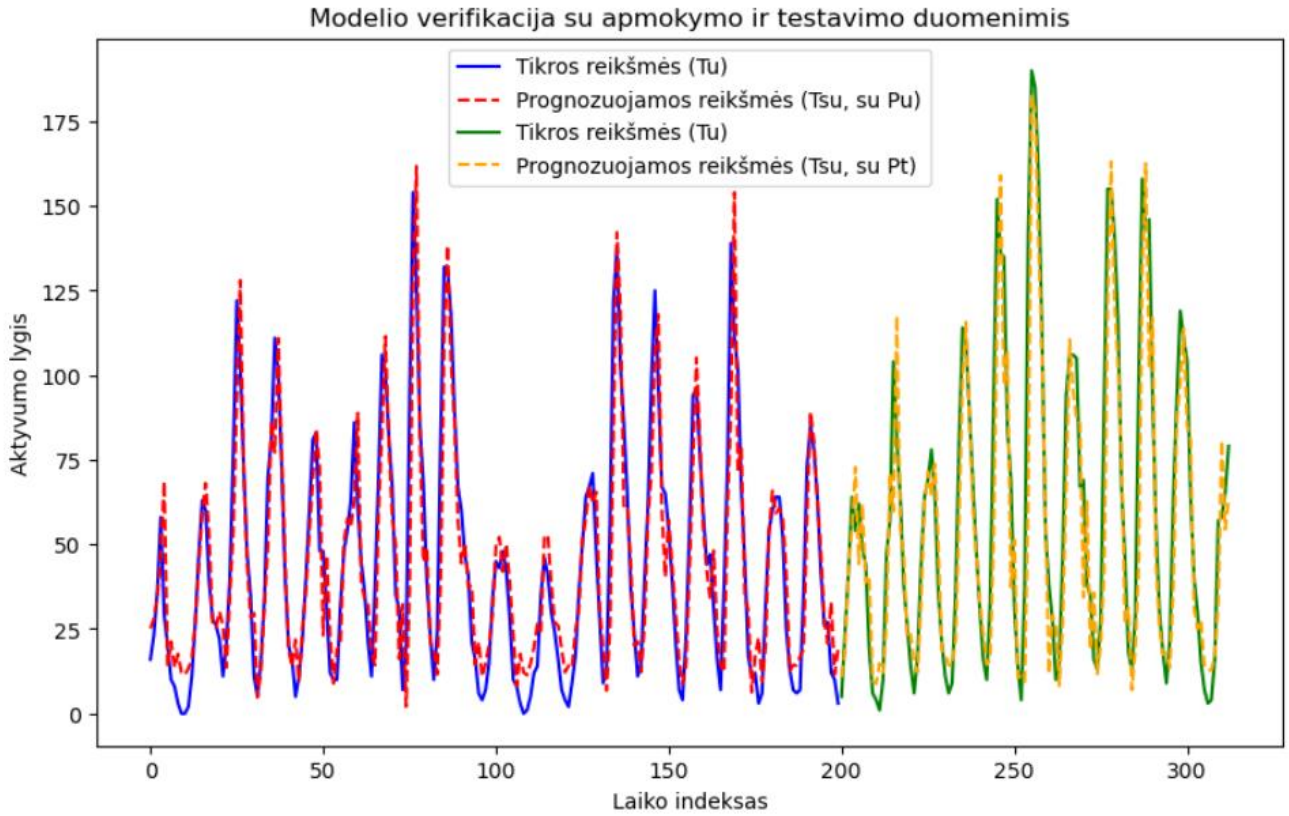


3.3. Tiesinės autoregresijos keoficientų reikšmės



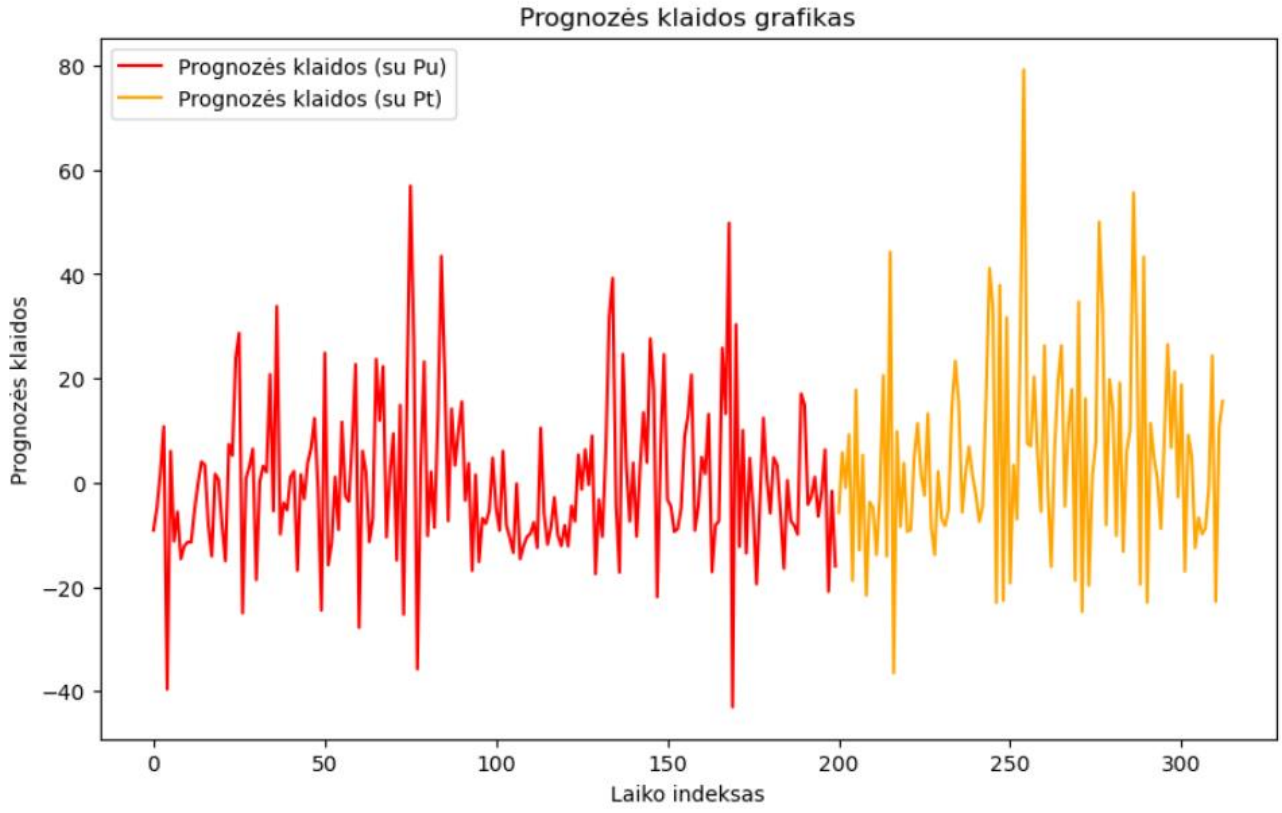
Modelio koeficientai (w1, w2, b): [-0.67608198 1.37150939] 13.403683236718116

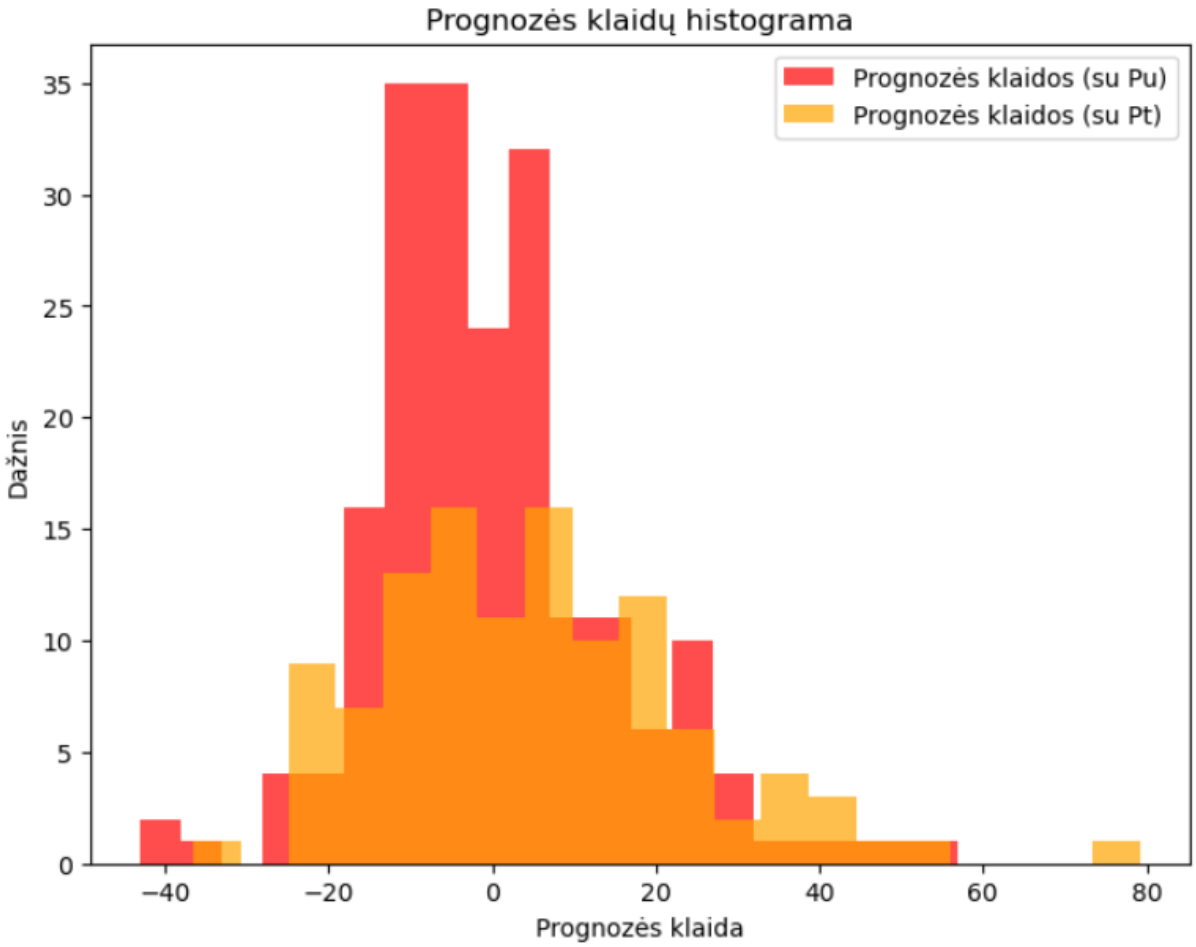
3.4. Modelio verifikacija



Prognozuojamos reikšmės yra gan panašios į tikrąsias, todėl modelis yra geras.

3.5. Prognozės klaidų grafikas bei histograma





3.6. MSE ir MDE

MSE su apmokymo duomenų rinkiniu (Pu): 217.1720799837935
MSE su testavimo duomenų rinkiniu (Pt): 386.40419314929693

MAD su apmokymo duomenų rinkiniu (Pu): 8.709692821703268
MAD su testavimo duomenų rinkiniu (Pt): 10.768026797610418

Iš MSE duomenų matome, kad modelis naudodamas apmokymo duomenų rinkinį duoda geresnius rezultatus, nei testavimo duomenimis. Tą patį galime pamatyti ir MAD duomenyse. Taigi, šie palyginimai rodo, kad modelis galbūt šiek tiek per daug prisitaikė prie apmokymo duomenų rinkinio, nes jis veikia geriau ant apmokymo duomenų lyginant su testavimo duomenimis.

3.7. Neuronai

Modelio svorio koeficientai:
[0.02501411 0.06294911 0.10144713 0.06211782 0.00485831 -0.00450769
-0.0061103 0.10486303 -0.18946514 -0.31495366 1.16050474]
MSE: 204.7484648436556
MAD: 7.690210484673706
Iš gautų duomenų matome, kad gavo geresnius rezultatus lyginant su rezultatais gautais panaudojant senesnius metodus.

3.8. Papildomi klausimai

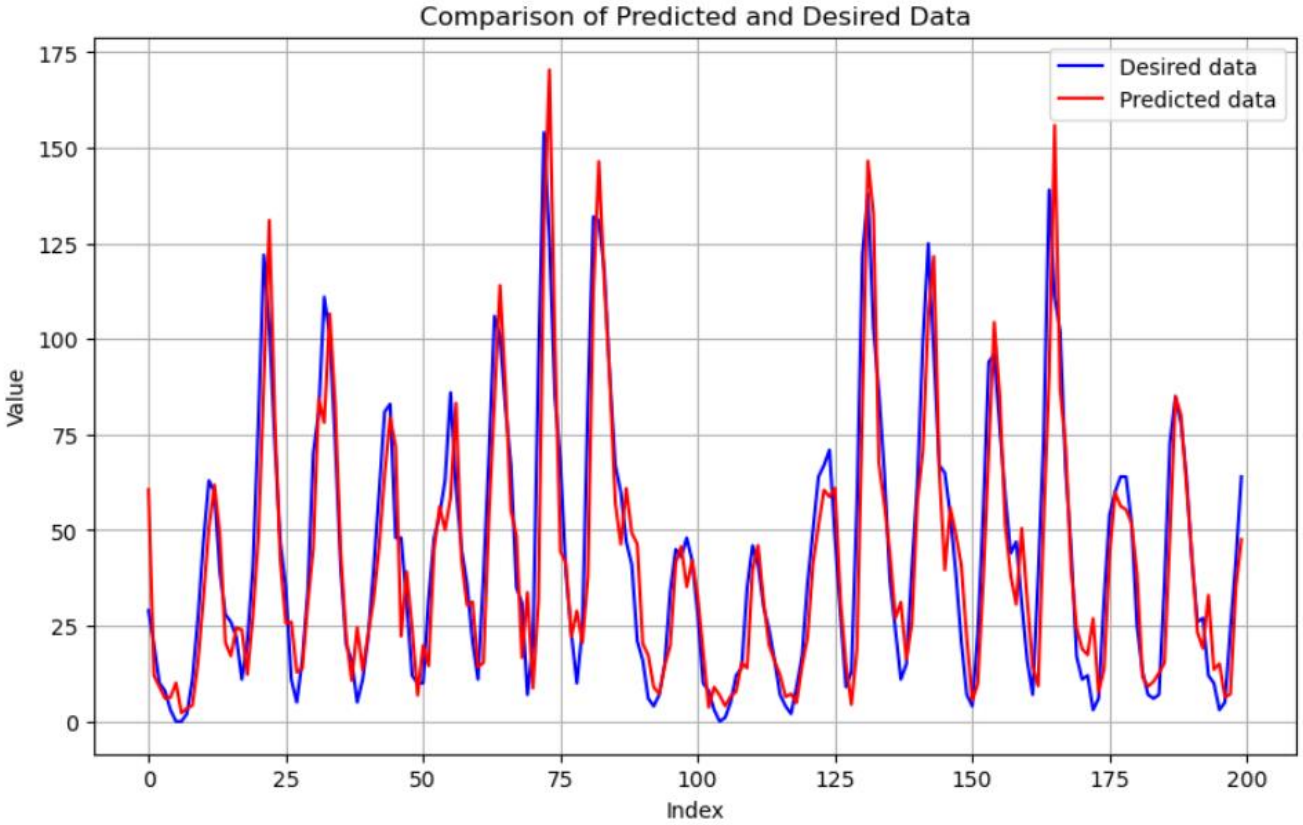
- **Ar mokymosi procesas yra konverguojantis? Jeigu ne, pamąstyti kas gali būti priežastimi ir pakeisti atitinkamą parametrą**. – Taip, procesas koverguoja, nes didinant maksimalų epochu kiekį matome, kad MSE ir MAD rezultatai keičiasi labai mažai.

- **Kokios yra naujos neurono svorių koeficientų reikšmės ?** –

  Modelio svorio koeficientai:
  [0.19458603 0.04440377 0.11073855 0.05063369 -0.0155591   0.07735881
   -0.13705856 0.15750634 -0.03512222 -0.58036385 1.30690148]

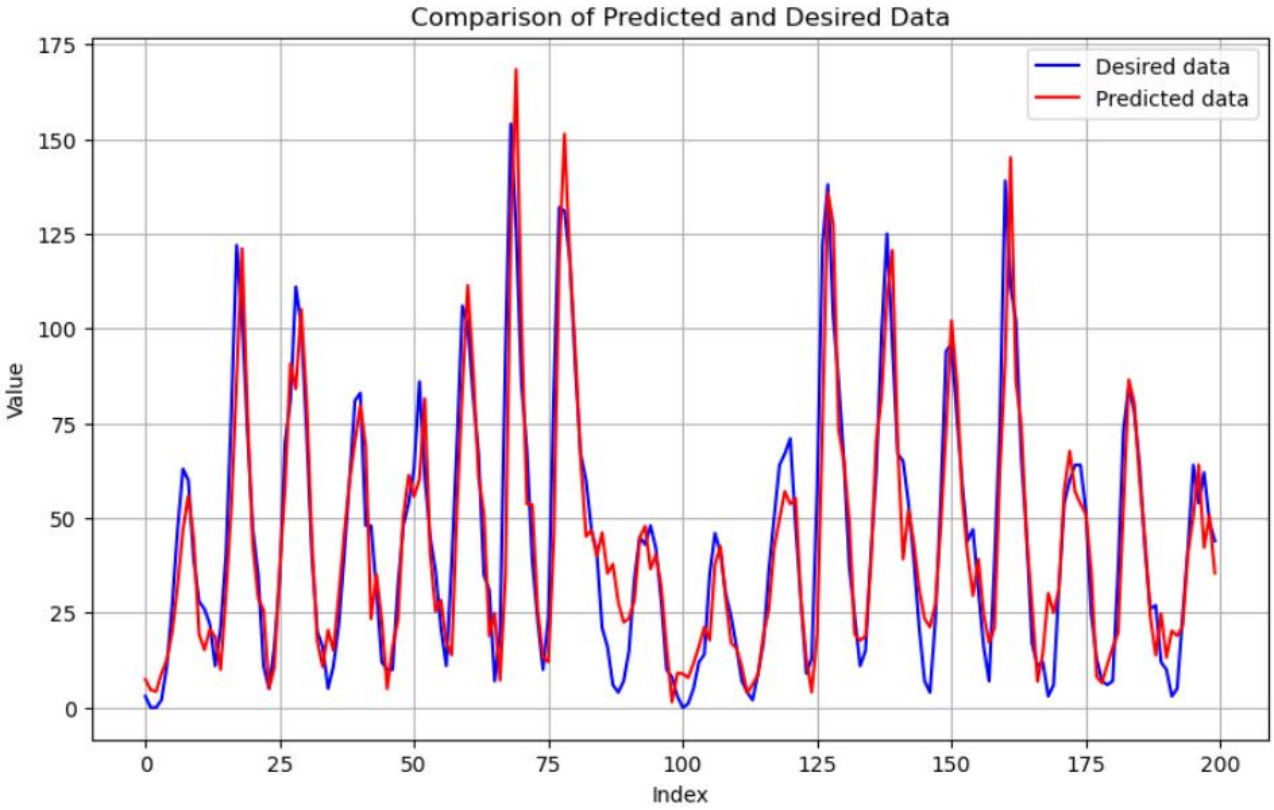
- **Kokia yra neurono darbo kokybės įverčio MSE ir MAD reikšmės ?**
  MSE: 198.20249811808634
  MAD: 7.715604370064284

3.9. N = 6



Modelio svorio koeficientai:
[0.05697369 0.22458603 -0.13451327 0.09022679 -0.17259468 -0.39955676
  1.34191379]
MSE: 243.49443018281818
MAD: 8.254555255385938

3.10. N = 10



Modelio svorio koeficientai:
[0.02501411 0.06294911 0.10144713 0.06211782 0.00485831 -0.00450769
 -0.0061103   0.10486303 -0.18946514 -0.31495366 1.16050474]
MSE: 204.7484648436556
MAD: 7.690210484673706

3.11. Kodas

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Nurodome failo kelią
file_path = "sunspot.txt"

# Nuskaitome duomenis, nustatydami, kad skirtukas yra tabuliavimo simbolis
df = pd.read_csv(file_path, sep="\t", header=None, names=["Metai", "Aktyvumas"])

# Filtruojame duomenis pagal metų intervalą
filtered_df = df[(df["Metai"] >= 1700) & (df["Metai"] <= 2014)]

# Nubrėžiame grafiką
plt.figure(figsize=(10, 6))
plt.plot(filtered_df["Metai"], filtered_df["Aktyvumas"], color='blue')
plt.title('Saulės dėmių aktyvumas (1700-2014 metai)')
plt.xlabel('Metai')
plt.ylabel('Aktyvumo lygis')
plt.grid(True)
plt.show()
```

```
# Sukuriame duomenų rinkinį
duomenys = filtered_df.values.tolist()

# Sukuriame mokymosi ir išvesties matricas
P = []
T = []

# Sukuriame mokymosi ir išvesties duomenų poras
for i in range(2, len(duomenys)):
    P.append([duomenys[i-2][1], duomenys[i-1][1]])  # Pridedame du ankstesnius metus
    T.append(duomenys[i][1])  # Pridedame atitinkamą išvesties duomenį

# Konvertuojame į numpy masyvus
P = np.array(P)
T = np.array(T)

# Sukuriame figūros ir ašių objektus
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Nubrėžiame įvesties duomenis
ax.scatter(P[:,0], P[:,1], T, c='r', marker='o')

# Pridedame ašių pavadinimus
ax.set_xlabel('Pirmas Įvesties Duomenys')
ax.set_ylabel('Antras Įvesties Duomenys')
ax.set_zlabel('Išvesties Duomenys')

# Pridedame diagramos pavadinimą
plt.title('Trimatis duomenų pasiskirstymas')

# Rodyti diagramą
plt.show()

# Išskiriame pirmus 200 įvesties ir išvesties duomenų fragmentus
Pu = P[:200]
Tu = T[:200]

from sklearn.linear_model import LinearRegression

# Sukurkime tiesinės regresijos modelį
modelis = LinearRegression()

# Apmokykime modelį naudojant apmokymo duomenų matricas Pu ir Tu
modelis.fit(Pu, Tu)

# Atspausdinkime modelio koeficientus
print("Modelio koeficientai (w1, w2, b):", modelis.coef_, modelis.intercept_)

# Gauti koeficientų reikšmes
w1, w2, b = modelis.coef_[0], modelis.coef_[1], modelis.intercept_

# Sukurti vardus koeficientams
koeficientai = ['w1', 'w2', 'b']
reiksmes = [w1, w2, b]
```

```
# Nubrėžiame barų diagramą
plt.bar(koeficientai, reiksmes, color ='maroon')

# Pridedame pavadinimus ir užrašą
plt.xlabel('Koeficientas')
plt.ylabel('Reikšmė')
plt.title('Koeficientų reikšmės')

# Rodyti diagramą
plt.show()

# Patikriname prognozes su apmokymo duomenų rinkiniu (Pu)
Tsu_pu = modelis.predict(Pu)

# Patikriname prognozes su testavimo duomenų rinkiniu (Pt)
Pt = P[200:]
Tt = T[200:]
Tsu_pt = modelis.predict(Pt)

# Nubraižome grafikus
plt.figure(figsize=(10, 6))

# Grafikas su apmokymo duomenų rinkiniu
plt.plot(Tu, label='Tikros reikšmės (Tu)', color='blue')
plt.plot(Tsu_pu, label='Prognozuojamos reikšmės (Tsu, su Pu)', color='red', linestyle='--')

# Grafikas su testavimo duomenų rinkiniu
plt.plot(range(200, len(T)), Tt, label='Tikros reikšmės (Tu)', color='green')
plt.plot(range(200, len(T)), Tsu_pt, label='Prognozuojamos reikšmės (Tsu, su Pt)', color='orange', linestyle='--')

# Pridedame pavadinimus ir legendą
plt.title('Modelio verifikacija su apmokymo ir testavimo duomenimis')
plt.xlabel('Laiko indeksas')
plt.ylabel('Aktyvumo lygis')
plt.legend()

# Rodyti grafiką
plt.show()

# Apskaičiuojame prognozės klaidų vektorių
e_pu = Tu - Tsu_pu
e_pt = Tt - Tsu_pt

# Nubraižome prognozės klaidų grafiką
plt.figure(figsize=(10, 6))

# Grafikas su apmokymo duomenų rinkiniu
plt.plot(e_pu, label='Prognozės klaidos (su Pu)', color='red')

# Grafikas su testavimo duomenų rinkiniu
plt.plot(range(200, len(T)), e_pt, label='Prognozės klaidos (su Pt)', color='orange')

# Pridedame pavadinimus ir legendą
plt.title('Prognozės klaidos grafikas')
plt.xlabel('Laiko indeksas')
plt.ylabel('Prognozės klaidos')
plt.legend()
```

```
# Rodyti grafiką
plt.show()

# Nubraižome prognozės klaidų histogramą
plt.figure(figsize=(8, 6))
plt.hist(e_pu, bins=20, color='red', alpha=0.7, label='Prognozės klaidos (su Pu)')
plt.hist(e_pt, bins=20, color='orange', alpha=0.7, label='Prognozės klaidos (su Pt)')
plt.title('Prognozės klaidų histograma')
plt.xlabel('Prognozės klaida')
plt.ylabel('Dažnis')
plt.legend()
plt.show()


# Apskaičiuojame MSE su apmokymo duomenų rinkiniu (Pu)
mse_pu = np.mean(np.square(e_pu))

# Apskaičiuojame MSE su testavimo duomenų rinkiniu (Pt)
mse_pt = np.mean(np.square(e_pt))

print("MSE su apmokymo duomenų rinkiniu (Pu):", mse_pu)
print("MSE su testavimo duomenų rinkiniu (Pt):", mse_pt)

# Apskaičiuojame absoliučias prognozės klaidas
abs_e_pu = np.abs(e_pu)
abs_e_pt = np.abs(e_pt)

# Apskaičiuojame MAD su apmokymo duomenų rinkiniu (Pu)
mad_pu = np.median(abs_e_pu)

# Apskaičiuojame MAD su testavimo duomenų rinkiniu (Pt)
mad_pt = np.median(abs_e_pt)

print("MAD su apmokymo duomenų rinkiniu (Pu):", mad_pu)
print("MAD su testavimo duomenų rinkiniu (Pt):", mad_pt)

import numpy as np

class AdaptiveLinearNeuron():
    def __init__(self, learning_rate=0.01, n_iter=50, error_goal=None):
        self.learning_rate = learning_rate
        self.n_iter = n_iter
        self.error_goal = error_goal

    def fit(self, X, y):
        self.w_ = np.zeros(1 + X.shape[1])
        self.cost_ = []

        for i in range(self.n_iter):
            output = self.net_input(X)
            errors = (y - output)
            self.w_[1:] += self.learning_rate * X.T.dot(errors)
            self.w_[0] += self.learning_rate * errors.sum()
            cost = (errors**2).sum() / 2.0
            self.cost_.append(cost)
```

```
            # Check if current error is below the desired error goal
            if self.error_goal is not None and cost < self.error_goal:
                print(f"Desired error goal reached at epoch {i+1}")
                break

        return self

    def net_input(self, X):
        return np.dot(X, self.w_[1:]) + self.w_[0]

    def activation(self, X):
        return self.net_input(X)

    def predict(self, X):
        return self.activation(X)

# Sukurkite savo duomenis ir taikykite modelį
X = Pu  # Įvesties duomenys
y = Tu  # Išvesties duomenys

lr = 0.0000001
error_goal = 225  # Siekiama mokymosi klaidos MSE reikšmė
max_epochs = 10000
  # Maksimalus epochų skaičius

model = AdaptiveLinearNeuron(learning_rate=lr, n_iter=max_epochs, error_goal=error_goal)
model.fit(X, y)

print("Modelio svorio koeficientai:")
print(model.w_)

# Predicted data
predicted = model.predict(X)

# Calculating MSE
mse = ((y - predicted) ** 2).mean()
print("MSE:", mse)

# Calculating MAD
absolute_errors = np.abs(y - predicted)
mad = np.median(absolute_errors)
print("MAD:", mad)

import matplotlib.pyplot as plt

# Predicted data
predicted = model.predict(X)

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(y, label='Desired data', color='blue')  # Desired data
plt.plot(predicted, label='Predicted data', color='red')  # Predicted data
plt.title('Comparison of Predicted and Desired Data')
plt.xlabel('Index')
plt.ylabel('Value')
plt.legend()
plt.grid(True)
```

```
plt.show()

import numpy as np

with open("sunspot.txt", 'r') as file:
    data = file.readlines()

matrix = [line.split() for line in data]

# Display the matrix
for row in matrix[:5]:
    print(row)

import matplotlib.pyplot as plt

# Išskirti metus ir saulės dėmių aktyvumą iš matricos
years = [int(row[0]) for row in matrix]
sunspot_activity = [float(row[1]) for row in matrix]

import numpy as np

# Sukurkime mokymosi duomenų matricą P
n = 10
P = []
for i in range(n, len(sunspot_activity)):
    P.append(sunspot_activity[i-n:i])

# Sukurkime išvesties duomenų matricą T
T = sunspot_activity[n:]

# Konvertuoti matricas į numpy masyvus
P = np.array(P)
T = np.array(T)


import numpy as np

# Apmokymo duomenų rinkinys
P_train = P[:200]
T_train = T[:200]

# Naujos matricos Pu ir Tu
Pu = P[:200]
Tu = T[:200]

# Sukuriame papildomą stulpelį su vienetiniais reikšmėmis
# Pridedame postūmį (bias)
P_train_with_bias = np.hstack((np.ones((P_train.shape[0], 1)), P_train[:, -(n-1):]))

# Apskaičiuojame svorio koeficientus naudodami normalinius lyginius kvadratus
# w = inv(X'X)X'Y
X = P_train_with_bias
Y = T_train
weights = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(Y)

# Atvaizduojame svorio koeficientus
print("Neurono svorio koeficientai:")
```

```
print(weights)
import matplotlib.pyplot as plt

# Funkcija, kuri prognozuoja išvesties reikšmes naudojant modelio svorio koeficientus ir įvesties duomenis
def predict_output(inputs, weights):
    predictions = np.dot(inputs, weights)
    return predictions

# Funkcija, kuri atvaizduoja grafikus su tikromis ir prognozuojamomis reikšmėmis
def plot_predictions(actual, predicted, title):
    plt.plot(actual, label='Tikrosios reikšmės', color='blue')
    plt.plot(predicted, label='Prognozuojamos reikšmės', color='red')
    plt.title(title)
    plt.xlabel('Indeksas')
    plt.ylabel('Saulės dėmių aktyvumas')
    plt.legend()
    plt.show()

# Apmokymo duomenų rinkinio prognozavimas
T_train_predicted = predict_output(P_train_with_bias, weights)

# Testavimo duomenų rinkinio prognozavimas
P_test = P[200:]
P_test_with_bias = np.column_stack((np.ones(len(P_test)), P_test[:, -(n-1):]))
T_test_predicted = predict_output(P_test_with_bias, weights)
T_test_actual = T[200:]
plot_predictions(T_test_actual, T_test_predicted, 'Prognozavimo rezultatai su testavimo duomenų rinkiniu')
# Sukurti prognozės klaidų vektorių
errors = T_test_actual - T_test_predicted

# Nubraižyti prognozės klaidų grafiką
plt.plot(errors, color='blue')
plt.title('Prognozės klaidų grafikas')
plt.xlabel('Indeksas')
plt.ylabel('Klaida')
plt.show()
# Nubraižyti prognozės klaidų histogramą
plt.hist(errors, bins=20, color='blue', alpha=0.7, edgecolor='black')  # Nustatome 20 stulpelių histogramoje
plt.title('Prognozės klaidų histograma')  # Pavadinimas
plt.xlabel('Klaida')  # X ašies pavadinimas
plt.ylabel('Dažnumas')  # Y ašies pavadinimas
plt.show()

# Apskaičiuoti vidutinės kvadratinės prognozės klaidos reikšmę (MSE)
MSE = np.mean(errors ** 2)

# Apskaičiuoti medianą absoliutaus nuokrypio (MAD)
MAD = np.median(np.abs(errors))

# Atvaizduoti rezultatus
print("Vidutinės kvadratinės prognozės klaida (MSE):", MSE)
print("Medianos absoliutus nuokrypis (MAD):", MAD)

import numpy as np

# Sukurkime mokymosi duomenų matricą P
n = 10
```

```
P = []
for i in range(n, len(sunspot_activity)):
    P.append(sunspot_activity[i-n:i])

# Sukurkime išvesties duomenų matricą T
T = sunspot_activity[n:]

# Konvertuoti matricas į numpy masyvus
P = np.array(P)
T = np.array(T)


import numpy as np

# Apmokymo duomenų rinkinys
P_train = P[:200]
T_train = T[:200]

class AdaptiveLinearNeuron():
    def __init__(self, learning_rate=0.01, n_iter=50, error_goal=None):
        self.learning_rate = learning_rate
        self.n_iter = n_iter
        self.error_goal = error_goal

    def fit(self, X, y):
        self.w_ = np.zeros(1 + X.shape[1])
        self.cost_ = []

        for i in range(self.n_iter):
            output = self.net_input(X)
            errors = (y - output)
            self.w_[1:] += self.learning_rate * X.T.dot(errors)
            self.w_[0] += self.learning_rate * errors.sum()
            cost = (errors**2).sum() / 2.0
            self.cost_.append(cost)

            # Patikriname, ar dabartinė klaida mažesnė nei norimas klaidos tikslas
            if self.error_goal is not None and cost < self.error_goal:
                print(f"Norima klaida pasiekiama po epochos {i+1}")
                break

        return self

    def net_input(self, X):
        return np.dot(X, self.w_[1:]) + self.w_[0]

    def activation(self, X):
        return self.net_input(X)

    def predict(self, X):
        return self.activation(X)

# Sukurkite savo duomenis ir taikykite modelį
X = P_train  # Įvesties duomenys
y = T_train  # Išvesties duomenys

lr = 0.0000001
```

```
error_goal = 225  # Siekiama mokymosi klaidos MSE reikšmė
max_epochs = 1000  # Maksimalus epochų skaičius

model = AdaptiveLinearNeuron(learning_rate=lr, n_iter=max_epochs, error_goal=error_goal)
model.fit(X, y)

print("Modelio svorio koeficientai:")
print(model.w_)

# Prognozuojamos duomenys
predicted = model.predict(X)

# Skaičiuojame MSE
mse = ((y - predicted) ** 2).mean()
print("MSE:", mse)

# Skaičiuojame MAD
absolute_errors = np.abs(y - predicted)
mad = np.median(absolute_errors)
print("MAD:", mad)
import matplotlib.pyplot as plt

# Predicted data
predicted = model.predict(X)

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(y, label='Desired data', color='blue')  # Desired data
plt.plot(predicted, label='Predicted data', color='red')  # Predicted data
plt.title('Comparison of Predicted and Desired Data')
plt.xlabel('Index')
plt.ylabel('Value')
plt.legend()
plt.grid(True)
plt.show()
```

4. 2 dalis

4.1. 10 intervalų kryžminės patikros eksperimentų rezultatai

10 intervalų kryžminės patikros eksperimentų rezultatai:
vidurkis = 0.21299999803304673
reikšmės =
[0.1899999976158142,
0.23999999463558197,
0.23999999463558197,
0.20000000298023224,
0.20000000298023224,
0.1599999964237213,
0.25,
0.20999999344348907,
0.20000000298023224,
0.23999999463558197]

Tikslinis atributas buvo “Weather”

Mokymosi greitis = 0.01

Buvo keičiamas mokymosi greitis ir epochu kiekis rezultatams pagerinti.

4.2. Kodas

```
import pandas as pd
import os
import matplotlib.pyplot as plt
from IPython.display import display
import numpy as np
from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

dp = pd.read_csv('futuristic_city_traffic.csv')
df = dp[:1000]

#display(df)  # This will display the DataFrame

weekday_mapping = {'Monday': 1, 'Tuesday': 2, 'Wednesday': 3, 'Thursday': 4, 'Friday': 5, 'Saturday': 6, 'Sunday': 7}
city_mapping = {'Ecoopolis': 1, 'MetropolisX': 2, 'Neuroburg': 3, 'SolarisVille': 4, 'TechHaven': 5, 'AquaCity': 6}
vehicle_mapping = {'Drone': 1, 'Car': 2, 'Flying Car': 3, 'Autonomous Vehicle': 4}
weather_mapping = {'Snowy': 1, 'Rainy': 2, 'Solar Flare': 3, 'Clear': 4, 'Electromagnetic Storm': 5}
economy_mapping = {'Stable': 1, 'Recession': 2, 'Booming': 3}
data = pd.DataFrame(df)
data['Day Of Week'] = data['Day Of Week'].map(weekday_mapping)
data['City'] = data['City'].map(city_mapping)
data['Vehicle Type'] = data['Vehicle Type'].map(vehicle_mapping)
data['Weather'] = data['Weather'].map(weather_mapping)
data['Economic Condition'] = data['Economic Condition'].map(economy_mapping)


display(data)

# Step 1: Split Data
X = data.drop(columns=['Weather'])
y = data['Weather']

# Step 2: Normalize Data
from sklearn.preprocessing import StandardScaler

# Define the learning rate
learning_rate = 0.01

# Initialize KFold cross-validation
kfold = KFold(n_splits=10, shuffle=True, random_state=42)

# Initialize list to store accuracy scores
accuracy_scores = []

# Perform cross-validation
for train_index, test_index in kfold.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
```

```
    y_train, y_test = y[train_index], y[test_index]

    # Build the model
    model = Sequential([
        Dense(7, activation='relu', input_shape=(X_train.shape[1],)),
        Dense(8, activation='relu'),
        Dense(1, activation='sigmoid')
    ])

    # Compile the model
    optimizer = Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])

    # Train the model
    model.fit(X_train, y_train, epochs=100, batch_size=100, verbose=0)

    # Evaluate the model on the test set and store the accuracy
    _, accuracy = model.evaluate(X_test, y_test, verbose=0)
    accuracy_scores.append(accuracy)

# Calculate the average accuracy
average_accuracy = np.mean(accuracy_scores)
print(f'Average accuracy across 10 folds: {average_accuracy}')
display(accuracy_scores)
```

5. Išvados

- Modeliai duoda gerus rezultatus, nes tikrosios reikšmės mažai skiriasi nuo prognozuojamų.
- Neuronų naudojimas pagerino rezultatus.
- Didesnis epochu kiekis nevisada duoda geresnius rezultatus.
- Modeliai veikimas eksponentiškai didėja.
- Modelių mokymosi tempo kitimas gali duoti geresnius rezultatus.