KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

Intelektikos pagrindai (P176B101)

Trečio laboratorinio darbo ataskaita

Atliko:

IFF-1/1 gr. Studentas

Vytenis Kriščiūnas

Priėmė:

lekt. Nečiūnas Audrius

lekt. Budnikas Germanas

KAUNAS 2024

TURINYS

1. Pirma dalis

Tikslas: susipažinti su prognozavimo uždavinio sprendimu panaudojant tiesinį dirbtinį neuroną, susipažinti su neuroninio tinklo mokymosi, testavimo ir jų panaudojimo uždaviniais.

1.1. Duomenų užkrovimas į darbinę atmintį

Duomenys yra nuskaitomi iš sunspot.txt failo ir išsaugomi df kintamajame.

```
text_file = "sunspot.txt"
df = pd.read_csv(text_file, delimiter='\t', header=None)
print(df)
```

Pirmame stulpelyje yra eilutės indeksai, antrame – metai ir trečiame – saulės dėmių reikšmės.

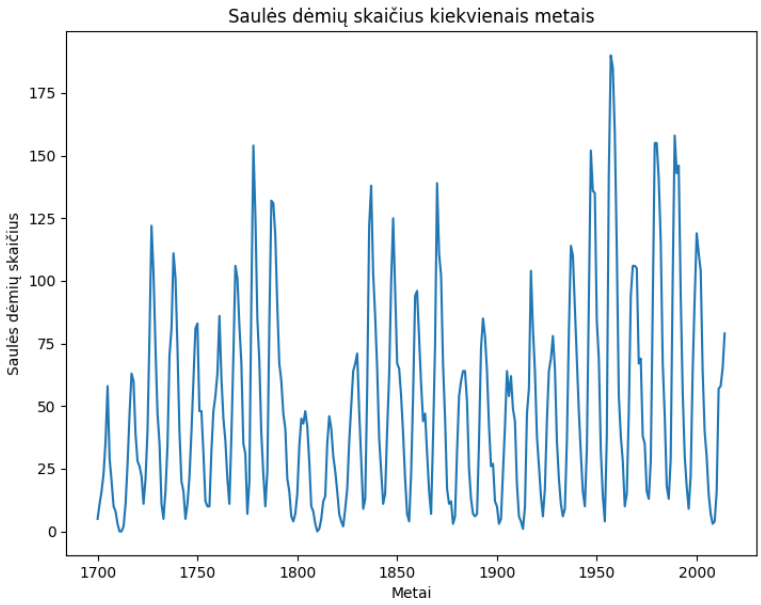
```
0    1700   5
1    1701  11
2    1702  16
3    1703  23
4    1704  36
..    ...  ..
310  2010  16
311  2011  57
312  2012  58
313  2013  65
314  2014  79
```

1 pav. Nuskaityti duomenys

1.2. Saulės dėmių aktyvumo 1700 – 2014 metų grafikas

Į x ir y kintamuosius yra atitinkamai išskiriami metai ir saulės dėmių skaičiai.

```
x = df.iloc[:, 0] #Metai
y = df.iloc[:, 1] #Sunspots
plt.plot(x, y)
plt.xlabel('Metai')
plt.ylabel('Saulės dėmių skaičius')
plt.title('Saulės dėmių skaičius kiekvienais metais')
plt.show()
```



2 pav. Saulės dėmių kiekvienais metais grafikas

1.3. Įvesties ir išvesties duomenų sudarymas (n=2)

Autoregresinio modelio eilė bus lygi n=2, tai reiškia, kad sekančių metų dėmių prognozė bus sudaroma iš dviejų ankstesnių metų dėmių. Neuronas turės du įėjimus ir vieną išėjimą.

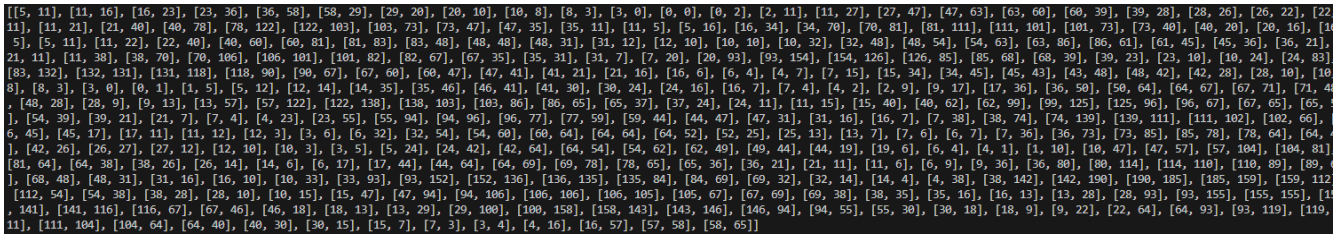
Reikia susidaryti P ir T atitinkamai: įvesties ir išvesties matricas.

```
P, T = devide_data(y, 2)
print(P)
print(T)
```

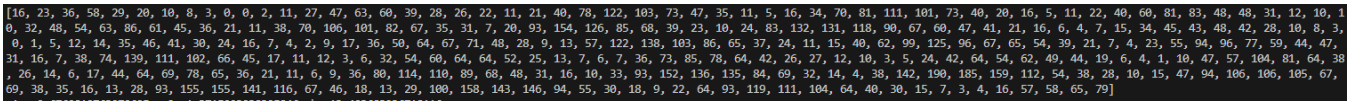
```
def devide_data(sunspots, n):
    P = [] #ivestis
    T = [] #isvestis

    for i in range(len(sunspots) - n):
        p_values = sunspots[i:i+n].tolist()
        P.append(p_values
        T.append(sunspots[i + n])

    return P, T
```



3 pav. Įvesties duomenys



4 pav. Išvesties duomenys

1.4. Trimatė įvesties ir išvesties diagrama

Reikia nubrėžti trimatę diagramą iš įvesties ir išvesties duomenų: x ašis – pirmo įėjimo reikšmę, y ašis – antrojo įėjimo reikšmnė ir z ašis – išejimo reikšmė.

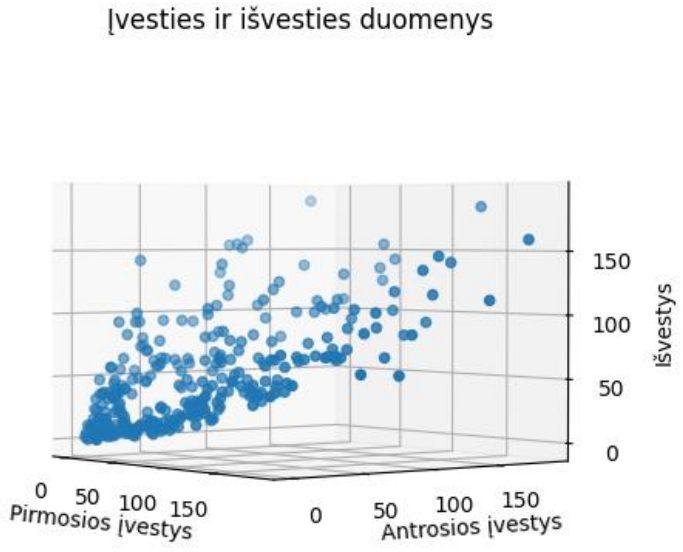
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

x = [p[0] for p in P]  # Pirmoji P value
y = [p[1] for p in P]  # Antroji P value
z = T                  # Isvestys

ax.scatter(x, y, z, marker='o')

ax.set_xlabel('Pirmosios įvestys')
ax.set_ylabel('Antrosios įvestys')
ax.set_zlabel('Išvestys')
plt.title('Įvesties ir išvesties duomenys')

plt.show()
```



5 pav. Trimatė įvesties ir išvesties diagrama

Sukiojant diagrama galima pastebėti tiesinę priklausomybę tarp duomenų – augant įvesties duomenų reikšmių dydžiams didėja ir išvesties dydžiai.

Grafinė interpretacija neurono svorių koeficientų yra plokštuma. Neurono svorio koeficientų optimalios reikšmės tūrėtų būti tokios, kad visų grafike esančių taškų atstumai būtų mažiausiai nutolę nuo plokštumos.

1.5. Apmokymo duomenų rinkinio išskyrimas

Reikia išskirti po 200 pradžioje esančių duomenų iš P ir T, taip sudarant du naujus duomenų rinkiniu Pu ir Tu, kurie bus naudojami apmokymui. Likę duomenys bus skirti modelio verifikavimui.

```
Pu, Tu = learn_data(P, T)

def learn_data(P, T):
    Pu = P[:200]
    Tu = T[:200]
    return Pu, Tu
```

1.6. Tiesinio autoregresijos modelio sukūrimas

Pasinaudojant užduotyje pateikta Python mokymosi medžiaga adresu: https://realpython.com/linear-regression-in-python/, bus kuriamas autoregresijos modelis.

```
from sklearn.linear_model import LinearRegression
```

```
X_train = np.array(Pu)
y_train = np.array(Tu)
model = LinearRegression()
model.fit(X_train, y_train)

w1 = model.coef_[0]
w2 = model.coef_[1]

b = model.intercept_

print(f"w1: {w1}, w2: {w2}, b: {b}")
```

```
w1: -0.6760819763970695, w2: 1.3715093938395846, b: 13.403683236718116
```

6 pav. Gauti autoregresijos modelio koeficientai

Iš gautų koeficientų galima sukurti jau anksčiau minėtą plokštumą trimatėje erdvėje.

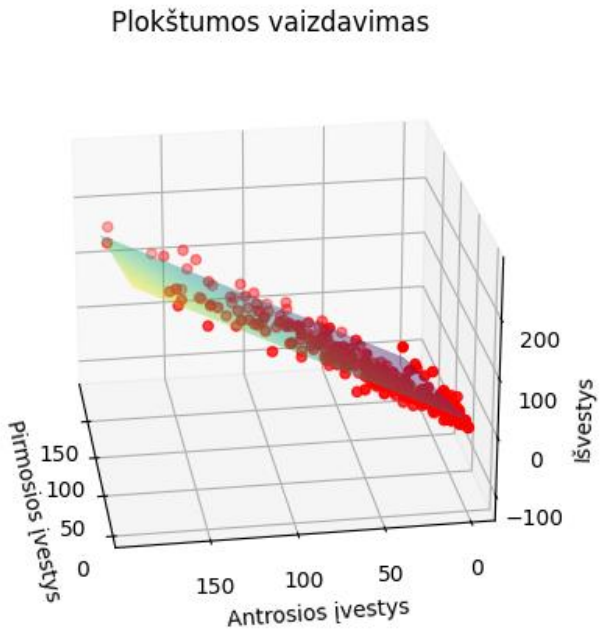
```
xx, yy = np.meshgrid(np.linspace(min(x), max(x)), np.linspace(min(y), max(y)))
zz = w1*xx + w2*yy + b
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

ax.scatter(x, y, z, color='r', label='Data points')

ax.plot_surface(xx, yy, zz, alpha=0.5, cmap='viridis')

ax.set_xlabel('Pirmosios įvestys')
ax.set_ylabel('Antrosios įvestys')
ax.set_zlabel('Išvestys')
ax.set_title('Plokštumos vaizdavimas')

plt.show()
```



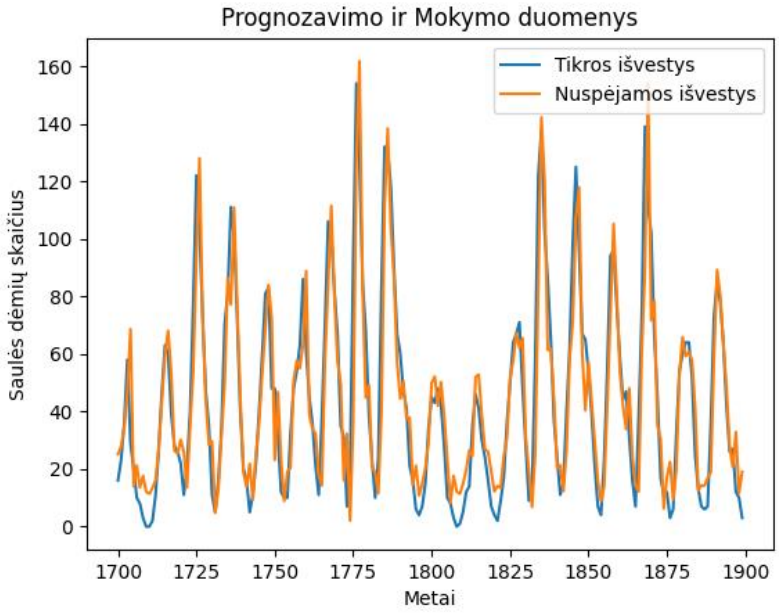
7 pav. Suformuota plokštuma pagal svorių koeficientus

1.7. Modelio verifikacijos grafikų sudarymas

Reikia verifikuoti gautą svorių koeficientų modelį su apmokymo duomenimis nuo 1702 – 1901 metų ir testavimo duomenimis nuo 1902 – 2014 metų. Tikrosios reikšmės – Tu ir prognozuojamos reikšmės – Tsu.

Testavimas su apmokymo duomenimis:

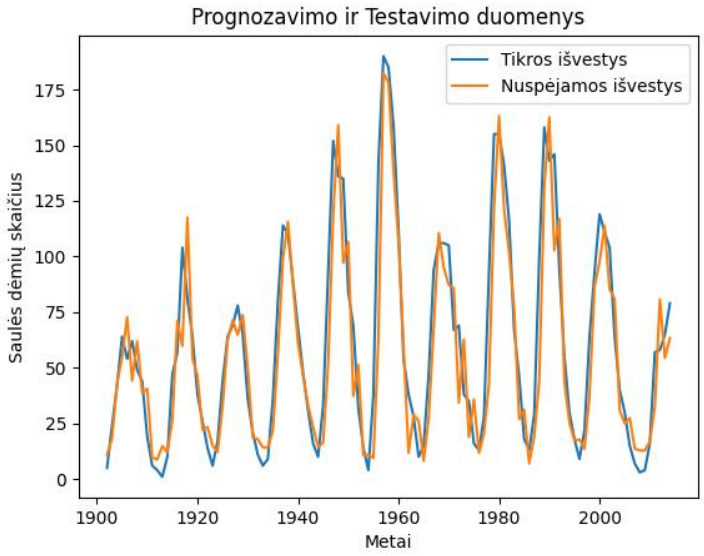
```
Tsu = model.predict(Pu)
years = np.array(df.iloc[:, 0])
plt.plot(years[:200], Tu, label='Tikros išvestys')
plt.plot(years[:200], Tsu, label='Nuspėjamos išvestys')
plt.xlabel('Metai')
plt.ylabel('Saulės dėmių skaičius')
plt.title('Prognozavimo ir Mokymo duomenys')
plt.legend()
plt.show()
```



8 pav. Testavimo su apmokymo duomenimis grafikas

Testavimas su testavimo duomenimis:

```
Pu_test, Tu_test = P[200:], T[200:]
Tsu_test = model.predict(Pu_test)
plt.plot(years[202:], Tu_test, label='Tikros išvestys')
plt.plot(years[202:], Tsu_test, label='Nuspėjamos išvestys')
plt.xlabel('Metai')
plt.ylabel('Saulės dėmių skaičius')
plt.title('Prognozavimo ir Testavimo duomenys')
plt.legend()
plt.show()
```



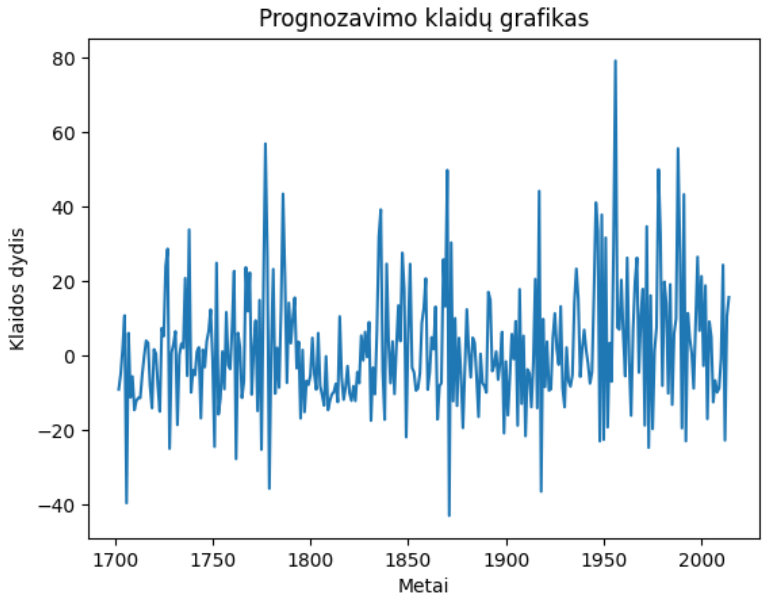
9 pav. Testavimo su testavimo duomenimis grafikas

1.8. Prognozės klaidos grafikas

Yra sukuriamas prognozės klaidos vektorius e ir nubraižomas 1700 – 2014 metų grafikas.

```
Ts = model.predict(P)
e = T - Ts

plt.plot(years[2:], e)
plt.xlabel('Metai')
plt.ylabel('Klaidos dydis')
plt.title('Prognozavimo klaidų grafikas')
plt.show()
```



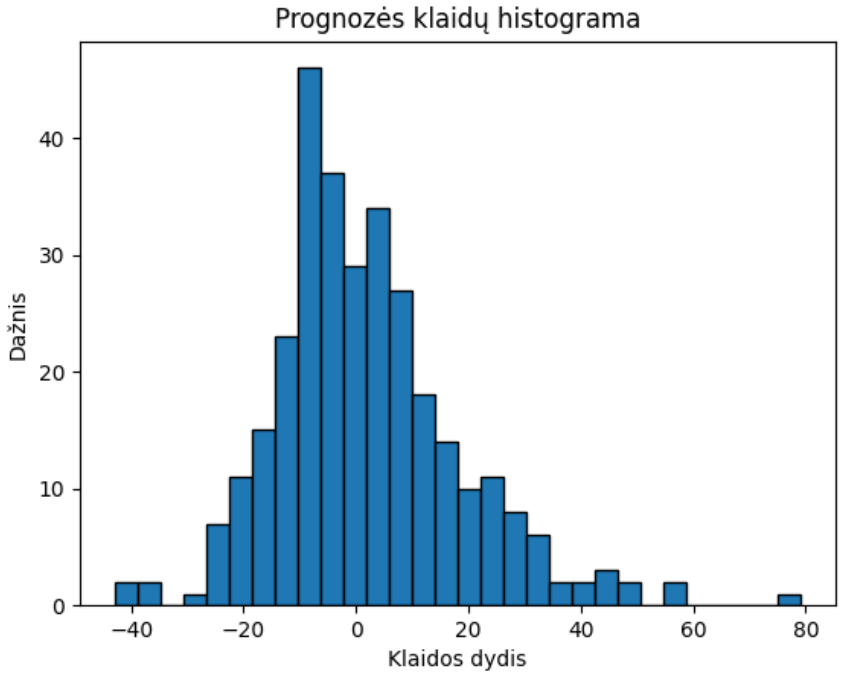
10 pav. Prognozavimo klaidų grafikas

Galima pastebėti, kad prognozavimo tikslumas yra gan įvairus, didžioji dalis klaidų yra išsidėstę intervale [0; 20].

1.9. Prognozės klaidų histograma

Pasitelkiant jau rastą klaidų vektoriu e yra braižoma histograma.

```
plt.hist(e, bins=30, edgecolor='black')
plt.xlabel('Klaidos dydis')
plt.ylabel('Dažnis')
plt.title('Prognozės klaidų histograma')
plt.show()
```



11 pav. Prognozės klaidų histograma

Atsižvelgus į histogramą galima matyti, kad didžiausia rasta klaida buvo lygi ~80, o dažniausia klaida ~10.

1.10. MSE ir MAD apskaičiavimas, palyginimas

Reikia rasti:

- MSE – vidutinės kvadratinės prognozės klaidos reiškmė;
- MAD – absoliutaus nuokrypio mediana.

```
MSE = np.mean(np.square(e))

MAD = np.median(np.abs(e))

print("Mean Squared Error (MSE):", MSE)
print("Median Absolute Deviation (MAD):", MAD)
```

```
Mean Squared Error (MSE): 278.2686575802851
Median Absolute Deviation (MAD): 9.218889655548999
```

12 pav. MSE ir MAD reikšmės

Gautų įverčių reikšmės stipriai skiriasi, nes MSE apskaičiavimui yra naudojamos visos rastos prognozės modelio klaidos, todėl dideli nuokrypiai nuo vidurkio daro stiprią įtaką galutiniai reikšmei. MAD atsižvelgia tik į vidurines modelio klaidų reikšmes, todėl anomalijos nėra įvertinamos.

1.11. Tiesinio neurono kūrimas

Svarbu nusistatyri lr žingsnio reikšmę kuri būtų intervale (0; 1]. Ciklo nutraukimo salygos bus: MSE pasiektos ribos dydis (pvz: MSE < 300) ir epochų kiekis (pvz: 100000).

Pasirinkau:

- lr: 0.000001;
- Epochų skaičių: 100000;
- MSE ribos dydis: 160;
- Pu įvesties ir Tu išvesties reikšmės yra apmokymo duomenys.

```
rez = fit(Pu, Tu, 100000, 0.000001)
print(rez)
```

```
def fit(X, Y, epoch_sk, lr):
    X = np.array(X)
    weight = np.zeros(1 + X.shape[1])
    errors = []
    costsMSE = []
    costsMAD = []

    for i in range(epoch_sk or MSE < 160):
        output = net_input(weight, X)
        errors = Y - output
        weight[1:] += lr * X.T.dot(errors)
        weight[0] += lr * errors.sum()

        MSE = (errors**2).sum() / len(Y)
        costsMSE.append(MSE)

        MAD = np.median(np.abs(errors))
```

```
        costsMAD.append(MAD)

        print(MSE)
    return weight

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

Atsakymai į klausimus:

- ✓ Mokymo procesas yra konverguojantis, nes pasirinkau pakankamai mažą žingsnio parametro reikšmę – lr. Jei pasirinktas žingsnis yra per didelis, procesas diverguotų, nepavyktų teisingai žengti priešinga gradientui kryptimi ir MSE reikšmės taptų begalinės.
- ✓ Naujos koeficientų reikšmės: w1 = -0.676, w2 = 1.372, b = 13.393
- ✓ MSE = 217.172 ir MAD = 8.704

1.12. Gautų svorių palyginimas

Autoregresijos modelio koeficientų reikšmės:

```
w1: -0.6760819763970695, w2: 1.3715093938395846, b: 13.403683236718116
```

Tiesinio neurono modelio koeficientų reikšmės:

```
w1: -0.6760083789296282 w2: 1.3715840291404637 b: 13.39346705263622
```

Galima pastebėti, kad gautos reikšmės yra praktiškai identiškos.

1.13. Tiesinio neurono kūrimas su testavimo duomenimis

Visi parametrai išlieka tokie patys išskyrus Pu ir Tu pasirenkami testavimo duomenys: nuo 1902 – 2014 metų.

Gauti rezultatai:

```
w1: -0.7078592198022357, w2: 1.392135430196164, b: 19.190434040189977
w1: -0.7062466872294204 w2: 1.393980991478665 b: 18.868046541601196 , MSE: 359.2544696816485, MAD: 12.876845320897559
```

13 pav. Testavimo duomenų pritaikymas skirtingiems modeliams

Autoregresinio ir tiesinio neurono modelių koeficientų reikšmės beveik nesiskiria. Atsižvelgus į modelio prognozavimo kokybę, galima teigti, kad ji yra prastesnė nei naudojant apmokymo duomenimis. Testavimo duomenų yra per pus mažiau, todėl MSE ir MAD reikšmės yra didesnės – modėlis nesugeba atlikti tikslesnių prognozių.

Maksimalis leistina lr reikšmė išlieka: 0.000001.

1.14. Lentelių sudarymas

Gautas reikšmes su duomenų kiekiais n=2, n=6 ir n=10 galima pavaizduoti lentelėje. Tiesinio neurono apmokymo modelio epochų skaičius nesikeitė: 100000.

n = 2		
lr = 0.000001	AM	TN
b	13.403	13.393
w1	-0.676	-0.676
w2	1.371	1.371
MSEm	217.172	217.172
MADm	8.709	8.703
MSEv	386.404	386.425
MADv	10.768	10.763

n = 6		
lr = 0.0000001	AM	TN
b	12.487	3.339
w1	0.153	0.249
w2	-0.239	-0.275
w3	0.125	0.153
w4	-0.030	-0.004
w5	-0.642	-0.679
w6	1.351	1.447
MSEm	211.10	224.052
MADm	8.333	7.896
MSEv	381.013	396.360
MADv	11.593	12.722

n = 10		
lr = 0.0000001	AM	TN
b	8.110	1.697
w1	0.011	0.038
w2	0.114	0.111
w3	0.034	0.047
w4	-0.031	-0.019
w5	0.062	0.075
w6	-0.153	-0.140
w7	0.142	0.154
w8	-0.051	-0.037
w9	-0.574	-0.579
w10	1.268	1.299
MSEm	190.871	195.682
MADm	8.158	7.936
MSEv	311.104	308.480
MADv	10.538	10.633

AM – autoregresijos modelis

TN – tiesinis neuronas

MSEm – MSE apskaičiuotas su mokymosi duomenimis

MSEv – MSE apskaičiuotas su verifikavimo duomenimis

MADm - MAD apskaičiuotas su mokymosi duomenimis

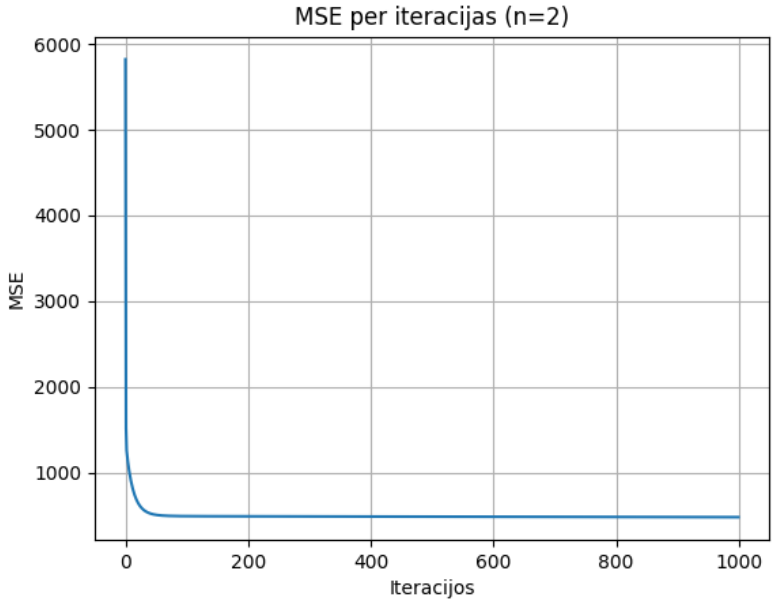
MADv - MAD apskaičiuotas su verifikavimo duomenimis

Pastebėjimai:

- Pagal gautus rezultatus galima teigti, kad didinant įėjimų (n) į neuroną kiekį MSE reikšmės tapo mažesnės, tai reiškia modelis prognozuoja tikslesnias išvestis.
- Didinant n kiekį, tenka keisti žingsnį (lr) – jį mažinti, kad modelis nediverguotų.
- Pasirinktus per didelį epochų arba n skaičių mokymosi metu gali įvykti persimokymas ir verifikavimo rezultatai bus prastesni.
- Apmokymui naudojant didesnius duomenų rinkinius yra gaunamos daug tikslenės prognozės – MSE stipriai sumažėja.
- Atliekant modelio verifikavimą su nežinomais duomenimis (testavimo) MSE reikšmė tampa žymiai prastesnė.
- Įėjimų (n), žingsnio (lr), epochų ir duomenų kiekio didinimas kainuoja laiką – apsimokymo procesas gali tampti tikslesnis, bet užtrunka ilgiau.

- Esant dideliam įėjimų skaičiui reiktų naudoti mažiau epochų arba atvirkščiai.

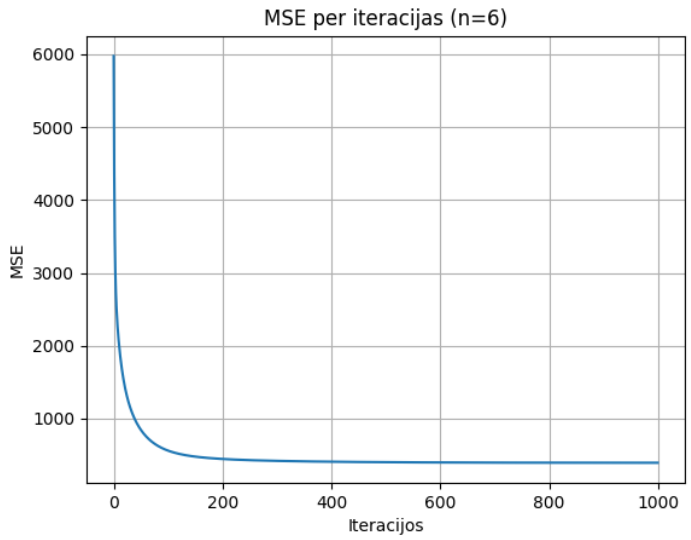
Keičiant apmokymo epochų kiekį galima rasti geriausią MSE reikšmę. Pasirinktus perdidelį epochų kiekį – įvyksta persimokymas, o pasirinkus per mažą – nedasimokymas. Maksimalus epochų skaičius: 1000. Tai pavaizdavau grafiškai:



14 pav. MSE pokytis su skirtingomis epochomis grafikas, kai n=2

```
Mažiausia MSE vertė: 480.6621754527466, epocha: 999.
```

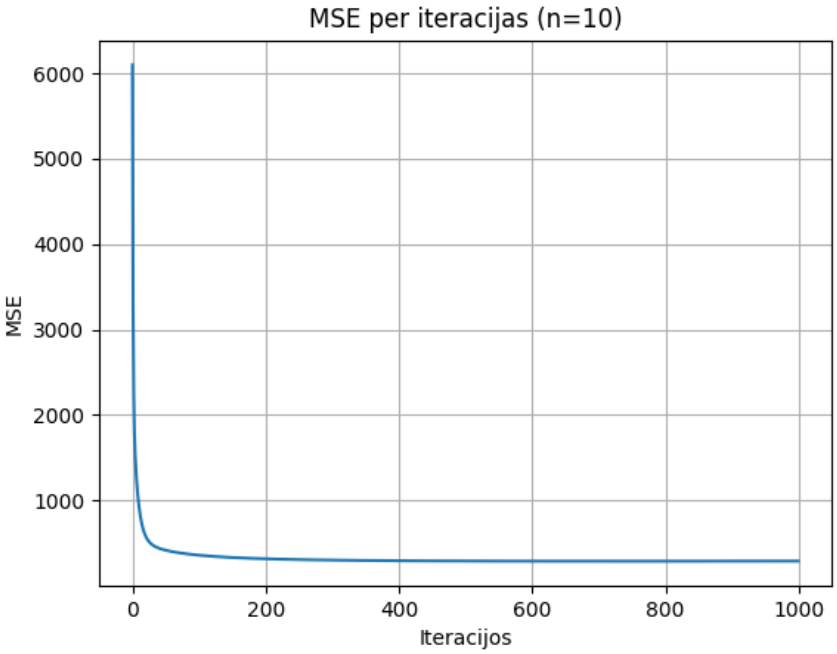
15 pav. Mažiausia MSE vertė, kai n=10



16 pav. MSE pokytis su skirtingomis epochomis grafikas, kai n=6

```
Mažiausia MSE vertė: 395.19785005888315, epocha: 964.
```

17 pav. Mažiausia MSE vertė, kai n=10



18 pav. MSE pokytis su skirtingomis epochomis grafikas, kai n=10

19 pav. Mažiausia MSE vertė, kai n=10

2. Antra dalis

Tikslas: pritaikyti įgytas žinias kuriant modelį prognozavimo ar klasifikacijos uždaviniui spręsti naudojant 1 laboratorinio darbo duomenų rinkinį.

2.1. Tikslo atributo pasirinkimas

Apie duomenis:

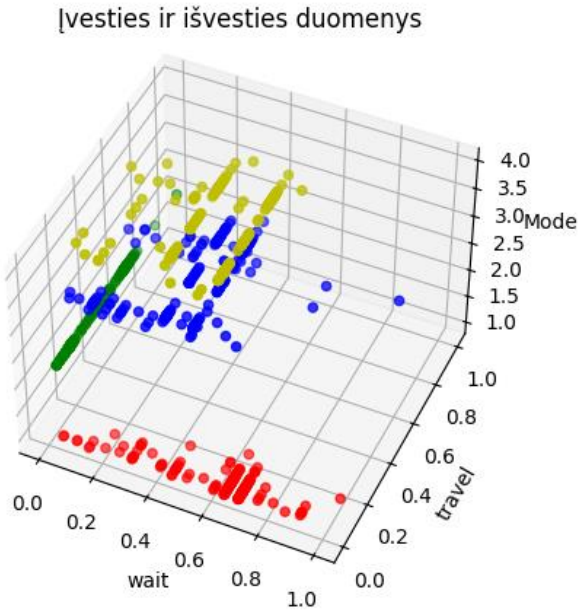
- „Individual“ – faktorius nurodantis individą nuo 1 iki 210 lygio;
- „Mode“ – faktorius indikuojantis kelionės rūšį: mašina, oru, traukiniu ar autobusu;
- „Choise“ – faktorius nurodantis pasirinkimą taip ar ne;
- „Wait“ – laukimo laikas terminale, 0 keliaujant mašina;
- „Vcost“ – transporto priemonės kaina;
- „Travel“ – kelionės trukmė transporto priemonėje;
- „Gcost“ – bendra kelionės kaina;
- „Income“ – uždarbis;
- „Size“ – žmonių kiekis.

Atributo pavadinimas	Kiekis (Eiluciu sk.)	Trukstamos reiksmes, %	Kardinalumas	Minimali reiksme	Maksimali reiksme	1-asis kvartilis	3-asis kvartilis	Vidurkis	Mediana	Standartinis nuokrypis
wait	840	0	26	0	99	1	53	34.58929	35	24.94861
vcost	840	0	135	2	180	23	67	47.76071	39	32.371
travel	840	0	405	63	1440	235	797	486.1655	397	301.4391
gcost	840	0	184	30	269	71	144	110.8798	102	47.97835
income	840	0	24	2	72	20	50	34.54762	35	19.67604

Atributo pavadinimas	Kiekis (Eiluciu sk.)	Trukstamos reiksmes, %	Kardinalumas	Moda	Modos daznumas	Moda, %	2-oji moda	2-osios modos daznumas	2-oji moda, %

mode	840	0	4	car	210	25	bus	210	25
choise	840	0	2	no	630	75	yes	210	25
size	840	0	6	Labai mazai	456	54.285 71	Mazai	232	27.619 05

Tikslo atributas bus *mode* – tai transporto priemonių pasirinkimo prognozavimas. Bus bandoma prognozuoti rementis *wait* ir *travel* stulpelių duomenimis.



20 pav Trimatis duomenų išsidėstymas pagal wait, travel ir mode

Galima pastebėti, kad skirtinų rūšių transporto priemonės yra gan neblogai susigrupavusios skirtingose grafiko dalyse, tai tik įrodo, kad duomenys yra susiję.

2.2. Duomenų pertvarkymas

Naudojami normalizuoti duomenys, kad išvengti labai didelių arba labai mažų reikšmių skaičiuojant sigmoidės funkciją.

Atributas *wait* turi šias galimas reikšmes: air, car, train ir bus. Šios kategorinės reikšės yra paverčiamos tolydinėmis dėl prognozavimo patogumo: air: 1, car: 2, train: 3 ir bus: 4.

```
file_path_learn = 'Normalizuoti_duomenys.csv'
df = pd.read_csv(file_path_learn)
```

```
# Define mapping dictionary
mode_mapping = {'air': 1, 'car': 2, 'train': 3, 'bus': 4}

# Replace categorical values with numerical values
df['mode'] = df['mode'].map(mode_mapping)

# Define features and target
features = ['wait', 'travel']
target = 'mode'


# Train data
X = df[features]
Y = df[target]
```

2.3. DNT architektūros schemos aprašymas

Mano naudojama DNT turi šią struktūra:

- Įvesties sluoksnis: 2 neuronai (atitinka įvesties požymius)
- Paslėptas sluoksnis: 4 neuronai (tiek yra galimų požymių), aktyvavimo funkcija – sigmoidė
- Išvesties sluoksnis: 1 neuronas, aktyvavimo funkcija – sigmoidė

2.4. Taikomas 10 intervalų kryžminės patikros metodas

Turimas duomenų failas yra išskaidomas į 10 dalių: 9 mokymo ir 1 testavimo. Apmokymas vyksta 10 kartų, taigi kiekviena dalis nors kartą yra naudojama kaip testavimo intervalas.

```
from sklearn.model_selection import KFold
```

```
# Sigmoid activation function and its derivative
def nonlin(x, deriv=False):
    if deriv:
        return x * (1 - x)
    return 1 / (1 + np.exp(-x))
# Define the number of folds
num_folds = 10

# Initialize KFold with 10 folds
```

```
kf = KFold(n_splits=num_folds)

# Initialize an empty list to store MSE values for each fold
mse_scores = []

# Iterate over each fold
for train_index, test_index in kf.split(X):
    # Split data into train and test sets for this fold
    X_train_fold, X_test_fold = X.iloc[train_index], X.iloc[test_index]
    y_train_fold, y_test_fold = Y.iloc[train_index], Y.iloc[test_index]

    # Define input and output data for this fold
    X_fold = X_train_fold.values
    y_fold = y_train_fold.values.reshape(-1, 1)

    # Initialize weights randomly with mean 0, 2 ivestys
    syn0_fold = 2 * np.random.random((2, 1)) - 1

    # Training loop for this fold
    for iter in range(10000):
        # Forward propagation
        l0_fold = X_fold
        l1_fold = nonlin(np.dot(l0_fold, syn0_fold))

        # Error calculation
        l1_error_fold = y_fold - l1_fold

        # Error weighted delta calculation
        l1_delta_fold = l1_error_fold * nonlin(l1_fold, True)

        # Update weights
        syn0_fold += np.dot(l0_fold.T, l1_delta_fold)

    # Make predictions on the test set for this fold
    l0_test_fold = X_test_fold.values
    l1_test_fold = nonlin(np.dot(l0_test_fold, syn0_fold))

    # Calculate MSE for this fold
    mse_fold = mean_squared_error(y_test_fold, l1_test_fold)

    # Append MSE to the list of scores
    mse_scores.append(mse_fold)

print(mse_scores)
# Calculate the average MSE over all folds
```

```
avg_mse = np.mean(mse_scores)

# Print the average MSE
print("Average Mean Squared Error (MSE) across 10 folds:", avg_mse)
```

```
[3.5000002410912106, 3.5000000000032916, 3.500000000001774, 3.5000002186348165, 3.500000000252895, 3.5000000003671863, 3.5000000000174842, 3.5000000000154627, 3.500000190769308, 3.500000003579292]
Average Mean Squared Error (MSE) across 10 folds: 3.500000065473272
```

21 pav. Sigmoidės tikslumo įverčiai ir vidutinis tikslumo įvertis

Akyvaizdu, kad tikslumo įvertis yra geras ir atskiri tikslumo įverčiai labai mažai skiriasi nuo vidutinio tikslumo įverčio.

2.5. DNT veiklos pagerinimas

Duomenų rinkinys yra normalizuotas, todėl jo keisti nereikia.

Bandžiau pakeisti aktyvacijos funkciją iš sigmoidės į ReLu, tačiau tikslumo įverčiai suprastėjo. Nors ši funkcija yra greitesnį už sigmoidę, jos tikslumas buvo prastesnis.

```
# Activation Function Modification Relu
def relu(x, deriv=False):
    if deriv:
        return np.where(x > 0, 1, 0)
    return np.maximum(0, x)
```

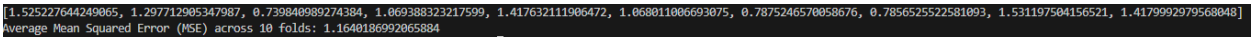
```
[7.5, 7.5, 7.5, 7.5, 7.5, 7.5, 7.5, 7.5, 7.5, 7.5]
Average Mean Squared Error (MSE) across 10 folds: 7.5
```

22 pav. ReLu tikslumo įverčiai ir vidutinis tikslumo įvertis

Pakeitus mokymosi greitį ir pritaikius ReLu funkciją pavyko pasiekti geresnių tikslumo įverčių rezultatų.

```
learning_rate = 0.001
```

```
 # Error weighted delta calculation
        l1_delta_fold = learning_rate * l1_error_fold * relu(l1_fold, True)
```



23 pav. Pasikeitę tikslumo įverčiai ir vidutinis tikslumo įvertis po mokymosi greičio pakeitimo

Pradinė modelio vidutinė tikslumo reikšmė buvo ~3.5, o pakeitus mokymosi greitį ~1.2. Taigi, pavyko pagerinti tikslumą net 65.71 %.

2.6. Išvados

- Pasirinkti duomenys DNT sudarymui turi būti tvarkingi – normalizuoti, be trūkstamų reikšmių;
- Kuo gaunamas MSE yra mažesnis tuo tikslenę prognozę atlieka DNT modelis.
- 10 intervalų kryžminės patikros metodas yra naudingas norint pilnai ištestuoti visus turimus duomenis ir gauti vidutinę MSE reikšmę.
- Modelio MSE reikšmės dydis priklauso nuo aktyvacijos funkcijos, mokymosi greičio, DNT truktūros ir naudojamų duomenų. Mano atvėju didžiausią naudą padarė naujos aktyvacijos funkcijos pasirinkimas ir mokymosi greičio sumažinimas.