KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FUKULTETAS

**Intelektikos pagrindai**

*(T120B029)*

**Laboratorinis darbas Nr. 2**

**Dirbtiniai neuroniniai tinklai**

Darbą atliko:  
IFF-7/14 gr. Studentas  
Eligijus Kiudys

Darbą priėmė:  
lekt. Andrius Nečiūnas  
doc. Agnė Paulauskaitė-Tarasevičienė

KAUNAS 2020

**Turinys**

[Įvadas 3](#_Toc38470568)

[Autoregresinis tiesinis modelis 4](#_Toc38470569)

[Uždavinio sprendimas panaudojant dirbtinį tiesinį neuroną 5](#_Toc38470570)

[Prognozavimo ir klasifikacijos modelių kūrimas 17](#_Toc38470571)

[Išvados 25](#_Toc38470572)

# Įvadas

**Tikslas**

Įsisavinti dirbtinių neuroninių tinklų (DNT) kūrimo, testavimo ir pritaikymo metodus.

**Uždaviniai:**

1. Susipažinti su dirbtinio neuroninio tinklo (DNT) apmokymu, testavimu bei panaudojimu;
2. Susipažinti su prognozavimo uždavinio sprendimu panaudojant tiesinį dirbtinį neuroną;
3. Pritaikyti įgytas žinias kuriant modelį prognozavimo ar klasifikacijos uždaviniui spręsti.

# Autoregresinis tiesinis modelis

Modelis, kuris realizuojamas esant prielaidai, kad priklausomybė tarp prognozuojamos reikšmės ir prieš tai esančių n elementų, gali būti aprašyta tiesine funkcija, vadinamas n-tosios eilės **autoregresiniu tiesiniu modeliu**. Autoregresinio modelio užduotis - laiko eilutės k-tosios reikšmės a(k) prognozavimas panaudojant n ankstesnes reikšmes a(k-1), a(k-2), ..., a(k-n).

Tiesinės autoregresijos modelio išraiška turi šią formą:

Čia w1, w2, ..., wn ir b modelio parametrai, o – prognozuojama reikšmė sekančiame žingsnyje.

Mūsų darbe autoregresinį modelį atitiks dirbtinis tiesinis neuronas, į kurį padavinėsime prieš tai buvusias n reikšmių. Modelio parametrai, neurono svoriniai koeficientai, prognozuos atsakymą, o prognozuojama reikšmė gausime neurono išėjime.

Prognozavimo klaidą k-tajame žingsnyje galime apskaičiuoti pagal formulę:

Čia atitinka tikrąją reikšmę, o – prognozuojamą.

Turint istorinių duomenų rinkinį ieškosime optimalių autoregresinio modelio parametrų reikšmių. Tai reiškia, kad sieksime, jog prognozė, sugeneruota mūsų modelio pagalba, skirtus labai mažai nuo tikru rezultatų .

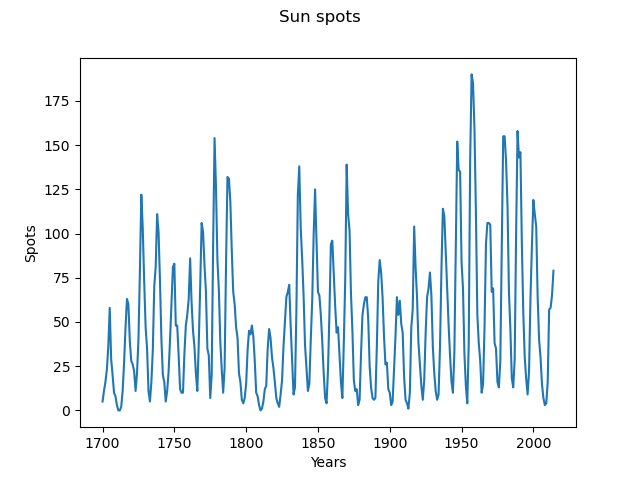
# Uždavinio sprendimas panaudojant dirbtinį tiesinį neuroną

**Užduotis:**

Sudaryti saulės dėmių skaičiaus prognozės modelį, kuris remtųsi n ankstesnių metų duomenimis. Uždavinį spręsti pasitelkiant paprasčiausios struktūros DNT – vienetinis neuronas su tiesine aktyvacijos funkcija.

Užkrauname duomenų rinkinio sunspot.csv turinį ir nubrėžiame saulės dėmių aktyvumo už 1700 – 2014 metus grafiką.

|  |
| --- |
| **import** numpy **as** np **import** Drawing **as** draw **from** sklearn **import** datasets, linear\_model, metrics   **def** ReadFile(fileName):  tempYear = []  sunSpotActivityTemp = []  file = open(fileName, **"r"**)  **for** line **in** file.readlines():  lineWithoutSpecial = line.strip()  split = lineWithoutSpecial.split(**"\t"**)  tempYear.append(int(split[0]))  sunSpotActivityTemp.append(int(split[1]))  **return** tempYear, sunSpotActivityTemp  year = [] sunSpotActivity = [] year, sunSpotActivity = ReadFile(**"sunspot.txt"**)  draw.DrowPlotSun(year, sunSpotActivity) |



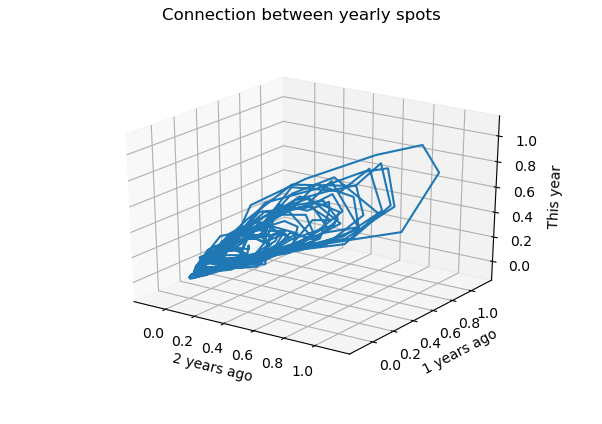
Pav 1. Saulės dėmių skaičiaus kitimo grafikas 1700-2014 metais

Priimdami, kad autoregresinio modelio eilė bus lygi 2 (n = 2), paruošiame neurono su dviem įvestimis mokymosi duomenų rinkinius P (įvesties duomenys) ir T (išvesties duomenys).

|  |
| --- |
| L = len(sunSpotActivity) sunSpotActivityDataUsage = [] **for** a **in** range(L-2):  sunSpotActivityDataUsage.append([int(sunSpotActivity[a]), int(sunSpotActivity[a+1])])  answerForSunActivity = [] answerForSunActivityGraphic = []  **for** element **in** sunSpotActivity[2:]:  answerForSunActivity.append([element])  answerForSunActivityGraphic.append(element) |

Įvesties ir išvesties duomenų rinkinių sąryšį pavaizduojame grafiškai (3D diagrama).

|  |
| --- |
| dataNormalized = normalizationFunction(sunSpotActivityDataUsage, min(sunSpotActivityDataUsage)[0], max(sunSpotActivityDataUsage)[0]) dataAnswNormalized = normalizationFunction(answerForSunActivity, min(answerForSunActivity)[0], max(answerForSunActivity)[0])   dataRes = [] dataRes.append([]) dataRes.append([]) **for** element **in** dataNormalized:  dataRes[0].append(element[0])  dataRes[1].append(element[1])  draw.DrowPlot3D(dataRes[0], dataRes[1], dataAnswNormalized)  **import** numpy **as** np **import** matplotlib.pyplot **as** plot  **from** mpl\_toolkits **import** mplot3d  **def** DrowPlot3D(x, y, z):  fig = plot.figure()  ax = fig.add\_subplot(111, projection=**'3d'**)  *# ax.plot3D(x, y, z)* ax.set\_xlabel(**'2 years ago'**)  ax.set\_ylabel(**'1 years ago'**)  ax.set\_zlabel(**'This year'**)  zline = np.array(x)  xline = np.array(y)  yline = np.array(z)  ax.plot3D(xline, yline, zline)  plot.suptitle(**'Connection between yearly spots'**)  plot.show() |



Pav 2. Grafinis įvesties ir išvesties duomenų rinkinių sąryšis

Optimalūs neurono svorio koeficientai aprašo lygtį plokštumos, geriausiai atitinkančios erdvės taškus.

Iš įvesties P ir išvesties T duomenų rinkinių išskiriame fragmentus Pu ir Tu, turinčius po 200 duomenų šablonų – apmokymo duomenų rinkinį. Remiantis šiuo duomenų rinkiniu apskaičiuosime optimalias neurono svorio koeficientų reikšmes (autoregresinio modelio parametrus).

|  |
| --- |
| dataForTraining = sunSpotActivityDataUsage[0:200] dataForTrainingAnswer = answerForSunActivity[0:200] |

Sukuriame dirbtinį neuroną ir apskaičiuojame jo svorio koeficientų reikšmes tiesioginiu metodu. Šiuo tikslu panaudojame apmokymo duomenų rinkinį Pu ir Tu.

|  |
| --- |
| **from** sklearn **import** linear\_model  net = linear\_model.LinearRegression() net.fit(np.array(dataForTraining), np.array(dataForTrainingAnswer)) |

Gauname neurono svorio koeficientų reikšmes.

|  |
| --- |
| w1 = net.coef\_[0][0] w2 = net.coef\_[0][1] b = net.intercept\_  print(**"Neurono Koficientai: "**) print(**"w1 = {}"**.format(w1)) print(**"w2 = {}"**.format(w2)) print(**"b = {}"**.format(b[0])) |

w1 = -0.6760819763970695

w2 = 1.3715093938395846

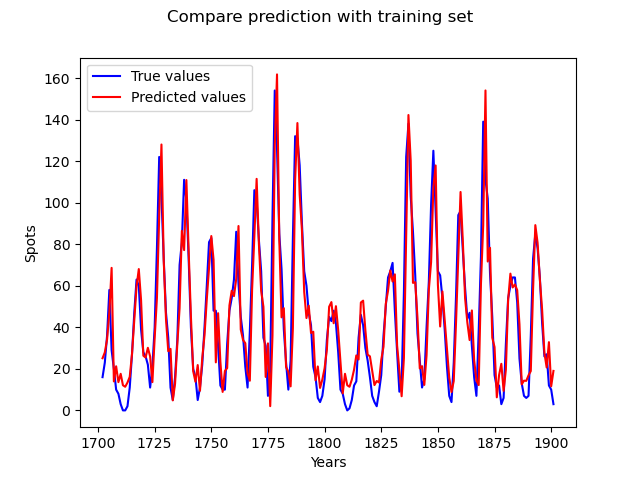
b = 13. 40368324

Atliekame modelio verifikaciją – patikriname prognozavimo kokybę atlikdami neurono veikimo imitaciją. Pirmiausiai verifikacijai pasitelkiame apmokymo duomenų rinkinį, kuris buvo panaudotas svorio koeficientams apskaičiuoti.

|  |
| --- |
| Tsu = net.predict(dataForTraining) |

Prognozavimo kokybę vertiname grafiškai sulygindami prognozės rezultatus ir tikrąsias reikšmes.

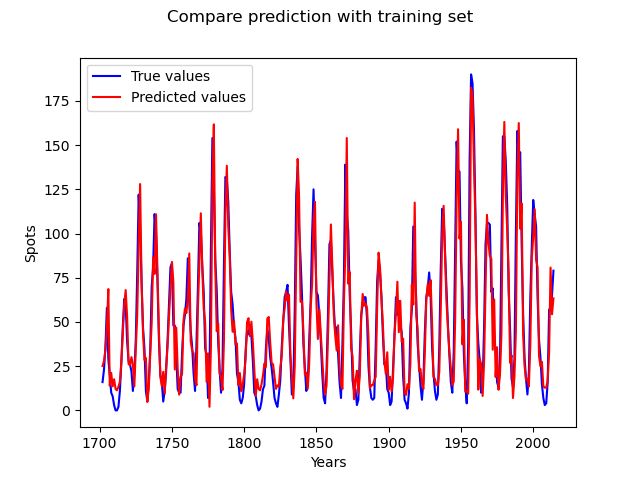
|  |
| --- |
| Tsu = net.predict(dataForTraining)  draw.DrawDiff(year[2:], sunSpotActivity[2:], Tsu)  **def** DrawDiff(Pu, Tu, Tsu):  plot.plot(Pu, Tu, **'b'**, label=**'True values'**)  plot.plot(Pu, Tsu, **'r'**, label=**'Predicted values'**)  plot.xlabel(**'Years'**)  plot.ylabel(**'Spots'**)  plot.suptitle(**'Compare prediction with training set'**)  plot.legend()  plot.show() |



Pav 3. Prognozuojamas ir tikrasis saulės dėmių skaičius 1702-1901 metais

Verifikaciją atliekame su visu duomenų rinkiniu.

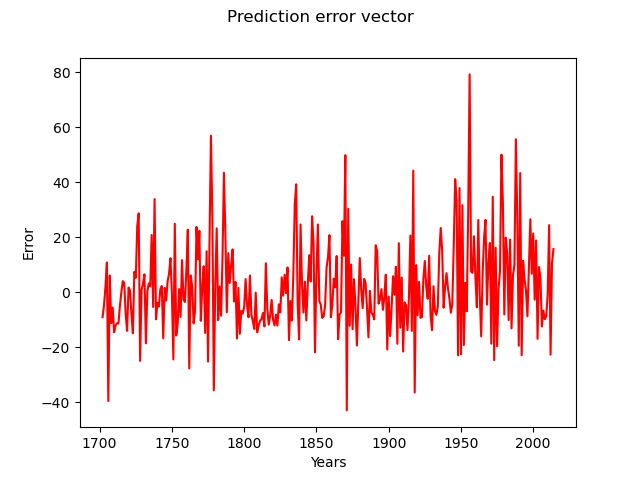
|  |
| --- |
| Ts = net.predict(sunSpotActivityDataUsage)  draw.DrawDiff(year[2:], sunSpotActivity[2:], Ts)  **def** DrawDiff(Pu, Tu, Tsu):  plot.plot(Pu, Tu, **'b'**, label=**'True values'**)  plot.plot(Pu, Tsu, **'r'**, label=**'Predicted values'**)  plot.xlabel(**'Years'**)  plot.ylabel(**'Spots'**)  plot.suptitle(**'Compare prediction with training set'**)  plot.legend()  plot.show() |



Pav 4. Prognozuojamas ir tikrasis saulės dėmių skaičius 1702–2014 metais (n = 2)

Sudarome prognozės klaidos vektorių bei jį pavaizduojame grafiškai.

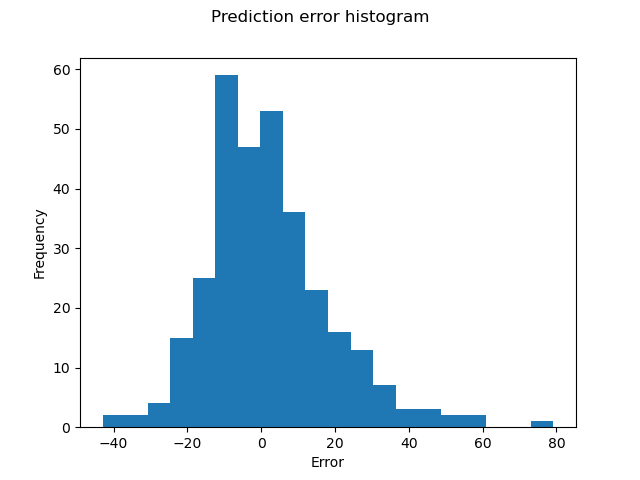
|  |
| --- |
| eVector = list() **for** real, predicted **in** zip(answerForSunActivity, Ts):  print(predicted)  e = real[0] - predicted[0]  eVector.append(e)  draw.DrowPlot(year[2:], eVector)  **def** DrowPlot(x, answ):  plot.plot(x, answ, **'r'**)  plot.xlabel(**'Years'**)  plot.ylabel(**'Error'**)  plot.suptitle(**'Prediction error vector'**)  plot.show() |



Pav 5. Saulės dėmių skaičiaus prognozavimo klaidos vektorius (n = 2)

Nubraižome prognozės klaidos histogramą.

|  |
| --- |
| draw.DrowHist(eVector)  **def** DrowHist(vect):  plot.hist(vect, bins=20)  plot.xlabel(**'Error'**)  plot.ylabel(**'Frequency'**)  plot.suptitle(**'Prediction error histogram'**)  plot.show() |



Pav 6. Saulės dėmių skaičiaus prognozės klaidų histograma (n = 2)

Apskaičiuojame vidutinės kvadratinės prognozės klaidos reikšmę (*eng. Mean-Square-Error* ***MSE***) ir prognozės absoliutaus nuokrypio medianą (*eng. Median-Absolute-Deviation,* ***MAD***).

**MSE** = 278.2687

**MAD** = 9.2189

**MSE** rodo paklaidos kvadratų vidurkį – tai gana gera metrika įvertinti visos prognozės tikslumą. Tuo tarpu **MAD** identifikuoja paklaidos reikšmę, apie kurią yra išsidėsčiusios paklaidos. **MAD** leidžia įvertinti prognozės paklaidą atmetant ekstremalias paklaidos reikšmes.

Toliau tą patį uždavinį sprendžiame sukurdami tiesinį neuroną, kuris svorinius koeficientus apskaičiuoja iteraciniu metodu.

Apibrėžiame maksimalų mokymosi žingsnių kiekį (*eng. Epochs*).

|  |
| --- |
| ep = 2000 |

Atliekame neurono apmokymą.

|  |
| --- |
| np.random.seed(1) syn0 = 2 \* np.random.random((2,1)) - 1 w\_initialize = syn0 bias = np.random.randn() b\_initialize = bias lr = 0.01 *# geriausias kai bias naudojamas lr yra 0.1* data = np.array(dataForTrainingNormalized) answerData = np.array(dataForTrainingAnswerNormalized) print(data) ep = 2000  **for** iter **in** range(ep):l0 = np.dot(data,syn0) + bias  l1 = calcDirivAndE(l0)  *# how much did we miss?* l1\_error = np.subtract(l1, answerData)  *# l1\_error = l1 - answerData  # print(l1\_error[0][0])* print(**"Error:"** + str(np.average(np.abs(l1\_error))))  *# multiply how much we missed by the  # slope of the sigmoid at the values in l1* l1\_delta = l1\_error \* calcDirivAndE(l1,**True**)  *# update weights and bias* syn0 = syn0 - lr \* np.dot(data.T, l1\_delta)  bias = bias - np.sum(lr \* l1\_delta) |

Patikriname, kokios neurono svorio koeficientų reikšmės nusistovėjo po apmokymo.

|  |
| --- |
| print(**'Neurono svoriniai koeficientai prieš apmokymą:'**) print(**"w1 = {}"**.format(w\_initialize[0])) print(**"w2 = {}"**.format(w\_initialize[1])) print(**"b = {}"**.format(b\_initialize)) print(**'----------'**) print(**'Neurono svoriniai koeficientai:'**) print(**"w1 = {}"**.format(syn0[0])) print(**"w2 = {}"**.format(syn0[1])) print(**"b = {}"**.format(bias)) |

Neurono svoriniai koeficientai prieš apmokymą:

w1 = -0.165955990594852

w2 = 0.4406489868843162

b = -0.5281717522634557

----------

Neurono svoriniai koeficientai:

w1 = -1.4991513019786302

w2 = 4.948811873011244

b = -2.042396766240324

Mokymosi procesas yra konverguojantis.

Atliekame saulės dėmių skaičiaus prognozę bei įvertiname jos paklaidą.

|  |
| --- |
| Ts = np.dot(dataNormalized, syn0) + bias TsRes = calcDirivAndE(Ts) TsDeNormalized = deNormalization(TsRes, min(answerForSunActivity)[0], max(answerForSunActivity)[0]) print(len(TsDeNormalized)) print(len(answerForSunActivity))  eVector = np.subtract(answerForSunActivity, TsDeNormalized)draw.DrowPlot(year[2:], eVector) draw.DrowHist(eVector)  predictionMSE = mse(eVector) predictionMAD = mad(eVector)  **def** mse(errors):  **if**(len(errors) == 0):  **return None** mseSum = 0  **for** e **in** errors:  mseSum += e\*e  **return** mseSum / len(errors)  **def** mad(errors):  **if**(len(errors) < 2):  **return None** absErrors = list(map(**lambda** x: abs(x), errors))  absErrors.sort()  index = int((len(absErrors) + 1) / 2)  **if**(len(absErrors) % 2 == 0):  result = absErrors[index - 1] + absErrors[index]  **return** result / 2  **else**:  **return** absErrors[index - 1]  print(**'MSE = {}'**.format(predictionMSE[0])) print(**'MAD = {}'**.format(predictionMAD[0])) |

**MSE** = 390.12060647

**MAD** = 13.28116649

**Epochs**

Eksperimentai rodo, jog **epochs** parametro reikšmę didinant iki 300, prognozavimo kokybė gerėja. Esant didesnėms parametro reikšmėms, prognozavimo kokybė didėja bet nežymiai.

**Learning-rate**

Eksperimentai rodo, jog maksimali **lr** parametro reikšmė lygi 0.01. Esant didesnėms parametro reikšmėms procesas nekonverguoja.

Toliau saulės dėmių skaičių prognozuojame ne pagal praėjusių 2 ankstesnių metų (n = 2), o pagal 10 ankstesnių metų (n = 10).

Patikriname neurono svorio koeficientų reikšmes prieš apmokymą ir po jo.

Neurono svoriniai koeficientai prieš apmokymą:

w0 = -0.165955990594852

w1 = 0.4406489868843162

w2 = -0.9997712503653102

w3 = -0.39533485473632046

w4 = -0.7064882183657739

w5 = -0.8153228104624044

w6 = -0.6274795772446582

w7 = -0.3088785459139045

w8 = -0.20646505153866013

w9 = 0.07763346800671389

b = 9

----------

Neurono svoriniai koeficientai:

w0 = -0.165955990594852

w1 = 0.4406489868843162

w2 = -0.9997712503653102

w3 = -0.39533485473632046

w4 = -0.7064882183657739

w5 = -0.8153228104624044

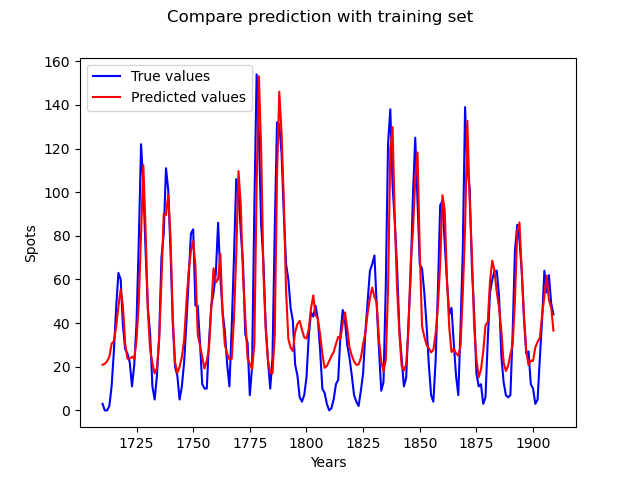
w6 = -0.6274795772446582

w7 = -0.3088785459139045

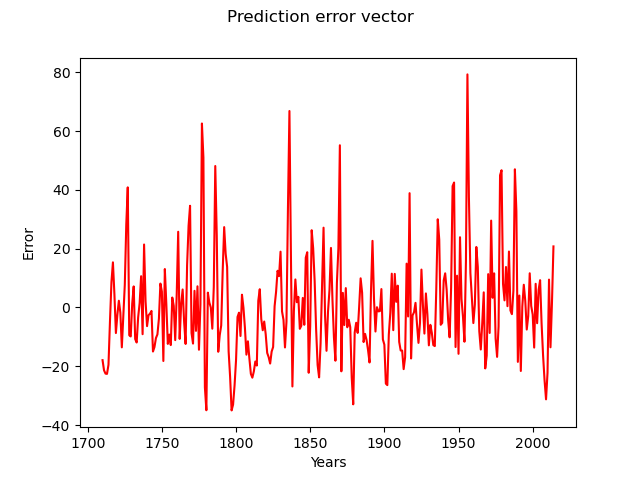
w8 = -0.20646505153866013

w9 = 0.07763346800671389

b = -2.030767208503895

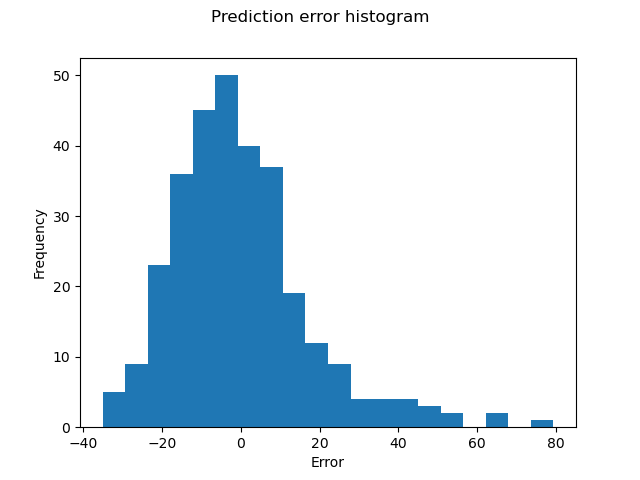
Atliekame neurono verifikaciją su visu duomenų rinkiniu

Pav 7. Prognozuojamas ir tikrasis saulės dėmių skaičius 1702–2014 metais (n = 10).

Sudarome prognozės klaidos vektorių bei jį pavaizduojame grafiškai.

Pav 8. Saulės dėmių skaičiaus prognozavimo klaidos vektorius (n = 10)

Nubraižome prognozės klaidos histogramą.



Pav 9. Saulės dėmių skaičiaus prognozės klaidų histograma (n = 10)

**MSE** = 303.40786860251933

**MAD** = 9.696342699852416

Tiek grafinė prognozės rezultatų analizė (7-9 pav.), tiek ir skaitinės charakteristikos (MSE ir MAD) rodo, jog remiantis 10 pastarųjų metų duomenimis galime padaryti geresnę prognozę nei darant pagal dviejų pastarųjų metų duomenis naudojant bibliotekas. Naudojant mano sukurtą modelį matome, kad rezultatai žymiai suprastėjo.

# Prognozavimo ir klasifikacijos modelių kūrimas

**Pasirinktas duomenų rinkinys:**

|  |  |
| --- | --- |
| Duomenų rinkinys | |
| Įrašų kiekis | 68785 |
| Atributų kiekis | 20 |
| Naudojamų atributų kiekis | 19 |
| Tolydinio tipo atributų kiekis | 17 |
| Kategorinio tipo atributų kiekis | 2 |
| Tolydus atributai | |
| Trukmė (angl. Duration) | |
| Plotis (angl. Width) | |
| Aukštis (angl. Height) | |
| Pralaidumas (angl. Bitrate) | |
| Kadrų dažnis (angl. Framerate) | |
| I (angl. I) | |
| P (angl. P) | |
| Kadrai (angl. Frames) | |
| I\_dydis (angl. I\_size) | |
| P\_dydis (angl. P\_size) | |
| Dydis (angl. Size) | |
| Išėigos pralaidimas (angl. O\_bitrate) | |
| Išeigos kadrų dažnis (angl. O\_framerate) | |
| Išeigos plotis (angl. O\_width) | |
| Išeigos aukštis (angl. O\_height) | |
| Sunaudotas RAM kiekis (angl. Umem) | |
| Užtruktas laikas (angl. Utime) | |
| Kategoriniai atributai | |
| Kodekas (angl. Codec) | |
| Konvertuotas Kodekas (angl. O\_codec) | |

Pasirinkau kaip rezultata Trukmė (angl. Duration) atributą, Ir bandau spėti su I ir Framerate atributais

Duomenų rinkinio fragmentas:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| id | duration | codec | width | height | bitrate | framerate | i | p | frames | i\_size |
| 04t6-jw9czg | 130.3567 | mpeg4 | 176 | 144 | 54590 | 12 | 27 | 1537 | 1564 | 64483 |
| 04t6-jw9czg | 130.3567 | mpeg4 | 176 | 144 | 54590 | 12 | 27 | 1537 | 1564 | 64483 |
| 04t6-jw9czg | 130.3567 | mpeg4 | 176 | 144 | 54590 | 12 | 27 | 1537 | 1564 | 64483 |
| 04t6-jw9czg | 130.3567 | mpeg4 | 176 | 144 | 54590 | 12 | 27 | 1537 | 1564 | 64483 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| p\_size | size | o\_codec | o\_bitrate | o\_framerate | o\_width | o\_height | umem | utime |
| 825054 | 889537 | mpeg4 | 56000 | 12 | 176 | 144 | 22508 | 0 |
| 825054 | 889537 | mpeg4 | 56000 | 12 | 320 | 240 | 25164 | 0.98 |
| 825054 | 889537 | mpeg4 | 56000 | 12 | 480 | 360 | 29228 | 1.216 |
| 825054 | 889537 | mpeg4 | 56000 | 12 | 640 | 480 | 34316 | 1.692 |

**Duomenų analizė:**

*Tolydinio tipo atributai*

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Atributo pavadinimas | Kiekis (Eilučių sk.) | Trūkstamos reikšmės, % | Kardinalumas | Minimali reikšmė | Maksimali reikšmė | 1-asis kvartilis | 3-asis kvartilis | Vidurkis | Mediana | Standartinis nuokrypis |
| duration | 68784 | 0 | 1086 | 31.08 | 25844.09 | 106.765 | 379.32 | 286.4139 | 239.1417 | 287.2556 |
| width | 68784 | 0 | 6 | 176 | 1920 | 320 | 640 | 624.9342 | 480 | 463.1657 |
| height | 68784 | 0 | 6 | 144 | 1080 | 240 | 480 | 412.5722 | 360 | 240.6137 |
| bitrate | 68784 | 0 | 1095 | 8384 | 7628466 | 134334 | 652967 | 693701.5 | 291150 | 1095620 |
| framerate | 68784 | 0 | 261 | 5.705752 | 48 | 15 | 29 | 23.24132 | 25.02174 | 7.224795 |
| i | 68784 | 0 | 306 | 7 | 5170 | 39 | 138 | 100.8683 | 80 | 84.76417 |
| p | 68784 | 0 | 1042 | 175 | 304959 | 2374 | 9155 | 6531.692 | 5515 | 6075.828 |
| frames | 68784 | 0 | 1044 | 192 | 310129 | 2417 | 9232 | 6641.708 | 5628 | 6153.298 |
| i\_size | 68784 | 0 | 1099 | 11648 | 90828552 | 393395 | 3392479 | 2838987 | 945865 | 4325105 |
| p\_size | 68784 | 0 | 1099 | 33845 | 7.69E+08 | 1851539 | 15155062 | 22180569 | 6166260 | 50972691 |
| size | 68784 | 0 | 1099 | 191879 | 8.07E+08 | 2258222 | 19773349 | 25022942 | 7881069 | 54143622 |
| o\_bitrate | 68784 | 0 | 7 | 56000 | 5000000 | 109000 | 3000000 | 1395036 | 539000 | 1749339 |
| o\_framerate | 68784 | 0 | 5 | 12 | 29.97 | 15 | 25 | 21.19086 | 24 | 6.668654 |
| o\_width | 68784 | 0 | 6 | 176 | 1920 | 320 | 1280 | 802.3364 | 480 | 609.9554 |
| o\_height | 68784 | 0 | 6 | 144 | 1080 | 240 | 720 | 503.8255 | 360 | 315.9681 |
| umem | 68784 | 0 | 9395 | 22508 | 711824 | 216820 | 219656 | 228224.7 | 219480 | 97430.17 |
| utime | 68784 | 0.004361 | 10960 | 0.184 | 224.574 | 2.096 | 10.433 | 9.996134 | 4.408 | 16.1076 |

Kategorinio tipo atributai

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Atributo pavadinimas | Kiekis (Eilučių sk.) | Trūkstamos reikšmės, % | Kardinalumas | Moda | Modos dažnumas | Moda, % | 2-oji Moda | 2-osios Modos dažnumas | 2-oji Moda, % |
| codec | 68784 | 0 | 4 | h264 | 31545 | 45.86096 | vp8 | 18387 | 26.73151 |
| o\_codec | 68784 | 0 | 4 | mpeg4 | 17291 | 25.13811 | vp8 | 17277 | 25.11776 |

Atliekame duomenų rinkinio pertvarkymą:

1. Pašalinamas atributas **id.**;
2. Pašalinami visi atributai, turintys reikšmių trūkumą didesnį kaip 60 proc.;
3. Atliekame tolydinio tipo atributų ekstremalių reikšmių korekciją;
4. Tuščias atributų reikšmes keičiame vidrukiu/moda;
5. Kategorinio tipo kintamuosius su Python paverčiau į tolydinio tipo kintamuosius naudodamas Dictionary mapping pvz. („MPEG4“=1, “H264”=2, “VP8”=3, “FLV”=4).
6. Naudojantis funkcija normaliztaion() atliekame įvesties atributų normalizavimą.

Sukuriame dirbtinį neruononį tinklą (DNT), sudarytą iš vieno neurono.

|  |
| --- |
| iDataNormalize = normalizationFunction(dataWithEmpty[**'i'**], min(dataWithEmpty[**'i'**]), max(dataWithEmpty[**'i'**])) framerateDataNormalize = normalizationFunction(dataWithEmpty[**'framerate'**], min(dataWithEmpty[**'framerate'**]), max(dataWithEmpty[**'framerate'**])) dataZip = zip(iDataNormalize, framerateDataNormalize) *# maybe use with all of this p* dataNormalized = list(dataZip) res = dataWithEmpty[**'duration'**] resData = [] **for** element **in** res:  resData.append([element]) resNormalized = normalizationFunction(dataWithEmpty[**'duration'**], min(dataWithEmpty[**'duration'**]), max(dataWithEmpty[**'duration'**]))  resDataNormalized = [] **for** element **in** resNormalized:  resDataNormalized.append([element]) *# print(dataNormalized[0:10])* np.random.seed(1) data = np.array(dataNormalized) answerData = np.array(resDataNormalized) *# fix answer data # print(dataSec)* folds = 10 dataSplitedForCross = np.array(np.array\_split(data, folds)) answSplitedForCross = np.array(np.array\_split(answerData, folds))  ep = 100 batch = 10  listOfErrors = [] **for** elements **in** range(folds):   trainData = []  trainRes = []  **if**(folds > 1):  cnt = 0  **for** merge **in** range(folds):  **if** (cnt > 0 **and** (merge < elements **or** merge > elements)):  trainData = np.concatenate((trainData, dataSplitedForCross[merge]), axis=0)  trainRes = np.concatenate((trainRes, answSplitedForCross[merge]), axis=0)   **if**(cnt == 0 **and** merge < elements):  trainData = dataSplitedForCross[0]  trainRes = answSplitedForCross[0]  cnt += 1  **elif**(cnt == 0 **and** merge > elements):  trainData = dataSplitedForCross[merge]  trainRes = answSplitedForCross[merge]  cnt += 1  **elif** (folds <= 1):  trainData = data  trainRes = answerData   syn0 = 2 \* np.random.random((2, 1)) - 1  w\_initialize = syn0  bias = np.random.randn()  b\_initialize = bias  lr = 0.01 *# geriausias kai bias naudojamas lr yra 0.1* batch\_Count = 0  **for** iter **in** range(ep): *# epochs  #  # forward propagation* l0 = np.dot(trainData, syn0) + bias *# privalo buti atskirta* l1 = calcDirivAndE(l0)  *# how much did we miss?* l1\_error = l1 - trainRes  *# print(l1\_error[0][0])* print(**"Error:"** + str(np.average(np.abs(l1\_error))))  *# multiply how much we missed by the  # slope of the sigmoid at the values in l1* l1\_delta = l1\_error \* calcDirivAndE(l1, **True**)  *# update weights* **if** batch\_Count == batch:  syn0 = syn0 - lr \* np.dot(trainData.T, l1\_delta)  bias = bias - np.sum(lr \* l1\_delta)  batch\_Count = 0  **elif** batch\_Count < batch:  batch\_Count += 1   l0 = np.dot(dataSplitedForCross[elements], syn0) + bias  l1 = calcDirivAndE(l0)  error = np.average(np.abs(answSplitedForCross[elements] - l1))  listOfErrors.append(error)   print(**'Neurono svoriniai koeficientai prieš apmokymą:'**) print(**"vidutinis tikslumas = {}"**.format(np.average(listOfErrors))) print(**"standartintis nuokrypit = {}"**.format(Deviation(listOfErrors)))  Ts = np.dot(dataNormalized, syn0) + bias TsRes = calcDirivAndE(Ts) TsDeNormalized = deNormalization(TsRes, min(dataWithEmpty[**'duration'**]), max(dataWithEmpty[**'duration'**]))  *# print(len(TsDeNormalized)) # print(len(answerForSunActivity)) #* eVector = list() **for** real, predicted **in** zip(answerData, TsDeNormalized):  e = real[0] - predicted[0]  eVector.append(e) *# # draw.DrowPlot(year[2:], eVector) # draw.DrowHist(eVector) #* predictionMSE = mse(eVector) predictionMAD = mad(eVector) print(**'MSE = {}'**.format(predictionMSE)) print(**'MAD = {}'**.format(predictionMAD)) |

**activation** = gradient

**loss** = paprasta atimtis

lr = 0.01

**batch\_size** = 10

**epochs** = 100

Atliekame 10 intervalų kryžminę patikrą.

|  |  |
| --- | --- |
| Nr. | Tikslumas |
| 1 | 0.008447905483476157 |
| 2 | 0.012776881680165046 |
| 3 | 0.007749118248355562 |
| 4 | 0.007365232564556326 |
| 5 | 0.009719445008227433 |
| 6 | 0.012584542192754171 |
| 7 | 0.007539158650896745 |
| 8 | 0.006085093118854678 |
| 9 | 0.00841685239046468 |
| 10 | 0.01823301132023518 |

**Kryžminės patikros rezultatai:**

vidutinis tikslumas = 0.009891724065798597

standartinis nuokrypis = 0.003470460168465797

MSE = 965.3517550339798

MAD = 31.071939657860845

Lentelėje pateikiamos išbandytos kiekvieno parametro reikšmės (viso 216 kombinacijų).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parametras** | **Išbandytos reikšmės** | | | |
| Batch size | 10 | 25 | 32 | 50 |
| Epochs | 100 | 500 | 1000 | 2000 |
| Lr | 0.0 | 0.1 | 0.01 | 0.001 |
| Folds | 1 | 10 | 10 | 10 |

|  |
| --- |
| np.random.seed(1) data = np.array(dataNormalized) answerData = np.array(resDataNormalized) *# fix answer data # print(dataSec)* folds = 10 ep = 1000 batch = 25 lr = 0.01 *# geriausias kai bias naudojamas lr yra 0.1* batch\_Count = 0  dataSplitedForCross = np.array(np.array\_split(data, folds)) answSplitedForCross = np.array(np.array\_split(answerData, folds))  listOfErrors = [] **for** elements **in** range(folds):   trainData = []  trainRes = []  **if**(folds > 1):  cnt = 0  **for** merge **in** range(folds):  **if** (cnt > 0 **and** (merge < elements **or** merge > elements)):  trainData = np.concatenate((trainData, dataSplitedForCross[merge]), axis=0)  trainRes = np.concatenate((trainRes, answSplitedForCross[merge]), axis=0)   **if**(cnt == 0 **and** merge < elements):  trainData = dataSplitedForCross[0]  trainRes = answSplitedForCross[0]  cnt += 1  **elif**(cnt == 0 **and** merge > elements):  trainData = dataSplitedForCross[merge]  trainRes = answSplitedForCross[merge]  cnt += 1  **elif** (folds <= 1):  trainData = data  trainRes = answerData   syn0 = 2 \* np.random.random((2, 1)) - 1  w\_initialize = syn0  bias = np.random.randn()  b\_initialize = bias  **for** iter **in** range(ep): *# epochs  #  # forward propagation* l0 = np.dot(trainData, syn0) + bias *# privalo buti atskirta* l1 = calcDirivAndE(l0)  *# how much did we miss?* l1\_error = l1 - trainRes  *# print(l1\_error[0][0])* print(**"Error:"** + str(np.average(np.abs(l1\_error))))  *# multiply how much we missed by the  # slope of the sigmoid at the values in l1* l1\_delta = l1\_error \* calcDirivAndE(l1, **True**)  *# update weights* **if** batch\_Count == batch:  syn0 = syn0 - lr \* np.dot(trainData.T, l1\_delta)  bias = bias - np.sum(lr \* l1\_delta)  batch\_Count = 0  **elif** batch\_Count < batch:  batch\_Count += 1   l0 = np.dot(dataSplitedForCross[elements], syn0) + bias  l1 = calcDirivAndE(l0)  error = np.average(np.abs(answSplitedForCross[elements] - l1))  listOfErrors.append(error)    print(**'Neurono svoriniai koeficientai prieš apmokymą:'**) print(**"vidutinis tikslumas = {}"**.format(np.average(listOfErrors))) print(**"standartintis nuokrypit = {}"**.format(Deviation(listOfErrors)))  print(listOfErrors)  Ts = np.dot(dataNormalized, syn0) + bias TsRes = calcDirivAndE(Ts) TsDeNormalized = deNormalization(TsRes, min(dataWithEmpty[**'duration'**]), max(dataWithEmpty[**'duration'**]))  *# print(len(TsDeNormalized)) # print(len(answerForSunActivity)) #* eVector = list() **for** real, predicted **in** zip(answerData, TsDeNormalized):  e = real[0] - predicted[0]  eVector.append(e) *# # draw.DrowPlot(year[2:], eVector) # draw.DrowHist(eVector) #* predictionMSE = mse(eVector) predictionMAD = mad(eVector) print(**'MSE = {}'**.format(predictionMSE)) print(**'MAD = {}'**.format(predictionMAD)) |

**Optimalios parametrų reikšmės:**

* Batch size = 32
* Epochs = 1000
* Learning(Lr) = 0.01
* Folds = 10

Atliekame pakeisto modelio 10 intervalų kryžminę patikrą.

|  |  |
| --- | --- |
| Nr. | Tikslumas |
| 1 | 0.008447905483476157 |
| 2 | 0.012776881680165046 |
| 3 | 0.007749118248355562 |
| 4 | 0.007365232564556326 |
| 5 | 0.009719445008227433 |
| 6 | 0.012584542192754171 |
| 7 | 0.007539158650896745 |
| 8 | 0.006085093118854678 |
| 9 | 0.00841685239046468 |
| 10 | 0.01823301132023518 |

**Kryžminės patikros rezultatai:**

vidutinis tikslumas = 0.00989163527370002

standartinis nuokrypis = 0.005290562461473264

MSE = 965.3517550339798

MAD = 31.071939657860845

# Išvados

1. Dirbtinių neuroninių tinklų (DNT) tinkamas naudojimas gali padėti išspręsti uždavinius, kurių sprendimo neįmanoma aprašyti alogritmiškai.
2. Teisingai panaudojant DNT, galima sudaryti prognozės modelį tarp iš pirmo žvilgsnio nepriklausomų atributų. Sudarant Čilės plebiscito visuomenės nuomonės apklausos klasifikacijos modelį, buvo pasiektas blogas tikslumas, nors atliekant duomenų analizę, sąryšis buvo didelis.
3. Didesnis DNT įvesčių skaičius („ilgesnis“ šablonas) gali padėti pasiekti geresnį prognozės rezultatą. Tai galime pastebėti sudarydami skirtingos eilės saulės dėmių skaičiaus prognozės modelį.
4. Didesnis mokymosi ciklų (*eng. Epochs*) skaičius nebūtinai sąlygoja geresnę apsimokymo kokybę – įvyksta persimokymas.
5. Kaip rodo vaido įrašų modelis, prognozės/klasifikacijos tikslumą galime pagerinti pataisan mokymosi konstantą ir eopochų dydį.
6. Optimalių modelio parametrų paieška (*eng. Paremeter Tunning*) gali padėti pasiekti dar didesnį modelio tikslumą. Tiesa pagerėjimas dažnai gali būti tik nežymus kaip ir matome paskutiniame punkte.
7. Neuroninis tinklas rašytas nuo nulio yra efektyvus kai datos kiekis yra mažas. Toks neuronins tinklas padeda suprasti kaip viskas veikias bet dažniausiai nėra efektyvus kadangi bibliotekos yra suoptimizuotos ir geriau bei greičiau veikia.