ՀԱՅԱՍՏԱՆԻ ՀԱՆՐԱՊԵՏՈՒԹՅԱՆ ԿՐԹՈՒԹՅԱՆ ԵՎ ԳԻՏՈՒԹՅԱՆ ՆԱԽԱՐԱՐՈՒԹՅՈՒՆ

ՀԱՅԱՍՏԱՆԻ ԱԶԳԱՅԻՆ ՊՈԼԻՏԵԽՆԻԿԱԿԱՆ ՀԱՄԱԼՍԱՐԱՆ

ՄԱԳԻՍՏՐՈՍԱԿԱՆ ԹԵԶ

ԹԵՄԱ՝ Ճանապարհատրանսպորտային խախտումների հայտնաբերումը մեքենայական ուսուցման կիրառմամբ

ՄԱԳԻՍՏՐԱՆՏ՝ Եղիգարյան Ռուբեն Ռոբերտի

ՄԱՍՆԱԳԻՏՈՒԹՅՈՒՆ՝ «Տեղեկատվական անվտանգություն»

ԿՐԹԱԿԱՆ ԾՐԱԳԻՐ՝ «Տեղեկատվական անվտանգություն»

ՈՐԱԿԱՎՈՐՄԱՆ ԱՍՏԻՃԱՆ՝ «Տեղեկատվական անվտանգության մագիստրոս» մագիստրոսի

ԵՐԵՎԱՆ 2020

ՀԱՍՏԱՏՄԱՆ ԹԵՐԹ

ԹԵՄԱ՝ Ճանապարհատրանսպորտային խախտումների հայտնաբերումը մեքենայական ուսուցման կիրառմամբ

Թեզի ղեկավար՝	ստորագրություն	Ռ. Գ. Հակոբյան տ.գ.թ. « .05.2020 »
Մագիստրանտ՝	ստորագրություն	Ռ.Ռ. Եղիգարյան « .05.2020 »
Գրախոս՝	ստորագրություն	Ա.Հ. Ազգանուն գիտական աստիճան և կոչում « .05.2020 »
Ամբիոնի վարիչ՝	ստորագրություն	Գ. Ի. Մարգարով տ.գ.թ., պրոֆեսոր « .05.2020 »
Ինստիտուտի տնօրեն՝	ստորագրություն	Ս.Ա. Մանուկյան տ.գ.թ., դոցենտ « .05.2020 »

ՏՎՅԱԼՆԵՐ ՇՐՋԱՆԱՎԱՐՏԻ ՄԱՍԻՆ

Մագիստրանտ՝	Եղիգարյան Ռուբեն Ռոբերտի
Մասնագիտություն՝	«Տեղեկատվական անվտանգություն»
Կրթական ծրագիր՝	«Տեղեկատվական անվտանգություն»
Ծննդյան տարեթիվը՝	1997
Մինչ մագիստրոսական որակավորումը՝	«Տեղեկատվական անվտանգություն» բակալավր
Մասնագիտությունը՝	«Տեղեկատվական անվտանգություն»
Կրթական ծրագիրը՝	«Տեղեկատվական անվտանգություն»
Հրատարակված աշխատանքներ՝	-

ՀԱՅԱՍՏԱՆԻ ԱԶԳԱՅԻՆ ՊՈԼԻՏԵԽՆԻԿԱԿԱՆ ՀԱՄԱԼՍԱՐԱՆ

ՏԵՂԵԿԱՏՎԱԿԱՆ և ՀԱՂՈՐԴԱԿՑԱԿԱՆ ՏԵԽՆՈԼՈԳԻԱՆԵՐԻ ՈՒ ԷԼԵԿՏՐՈՆԻԿԱԱՅԻ ԻՆՍՏԻՏՈՒՏ

Տեղեկատվական անվտանգության և ծրագրային ապահովման ամբիոն

Մասնագիտություն՝ Տեղեկատվական անվտանգություն, դասիչ՝ 061901.01.7

Կրթական ծրագիր՝ Տեղեկատվական անվտանգություն

Թիվ ՄՏՏ955 ակադեմիական խմբի

Եղիգարյան Ռուբեն Ռոբերտի
(ուսանողի ազգանուն անուն հայրանուն)

ՄԱԳԻՍՏՐՈՍԱԿԱՆ ԹԵԶԻ ԱՌԱՋԱԴՐԱՆՔ

1. Թեման՝ Ճանապարհատրանսպորտային խախտումների հայտնաբերումը մեքենայական ուսուցման կիրառմամբ

Հաստատված է ՀԱՊՀ 2016 թ.-ի հոկտեմբեր « 26 » թիվ 01-11/910 հրամանով

2. Նախնական տվյալներ

Python և Java ծրագրավորման լեզուներ, մեքենայական ուսուցման գրադարաններ Python լեզվի համար

3. Հաշվեբացատրագրի բովանդակություն (բաժինների և մշակման ենթակա հարցերի թվարկմամբ)

Գրականության նյութերի ամփոփում և խնդրի դրվածք, օգտագործվող մեթոդներ, ծրագրի նախագծում, ծրագրի իրագործում և կիրառում։

4. Թեզի կատարման օրացույցային պլան

Հ/Հ	Թեզի կատարման փուլերը			Ծանոթություն
	Անվանումը	Կատ. ժամկ.	հաշ. ձևը	
1.	Ճանապարհատրանսպորտային խախտումների գրանցման առկա համակարգերի ուսումնասիրություն	17.03.20	գրավոր	
2.	Հնարավոր ճանապարհատրանսպորտային խախտումների ուսումնասիրություն	07.04.20	գրավոր	
3.	Հնարավոր ճանապարհատրանսպորտային խախտումների ուսումնասիրություն	28.04.20	գրավոր	
4.	Խնդրի դրվածք	28.04.20	գրավոր	
	I ատեստավորում		40 %	
5.	Մեքենայական ուսուցման տարրեր	15.09.20	գրավոր	
6.	YOLO - իրական ժամանակում օբյեկտների հայտնաբերում	27.10.20	գրավոր	
7.	Համակարգի մանրամասն նկարագրություն	23.12.20	գրավոր	
	II ատեստավորում		70 %	
9.	Համակարգի ծրագրային իրականացման նկարագրություն	30.03.21	գրավոր	
10.	Ծրագրային ապահովման նախագծում	20.04.21	գրավոր	
	III ատեստավորում		100%	
12.	Աշխատանքի ներկայացումը ամբիոն	27.04.21	Ավ. աշխ.	
13.	Նախնական պաշտպանություն	04.05.21		

5. Աշխատանքի պաշտպանության օրը՝

6. Ամբիոնի վարիչ՝ Գ. Ի. Մարգարով

(Ա.Ա.Հ., ստորագրություն, ամսաթիվ)

7. Աշխատանքի ղեկավար Ռ.Գ. Հակոբյան

(Ա.Ա.Հ., ստորագրություն, ամսաթիվ)

8. Աշխատանքի առաջադրանքը ստացա Ռ.Ռ. Եղիգարյան

(ուսանողի Ա.Ա.Հ., ստորագրություն, ամսաթիվ)

ՀԱՄԱՌՈՏԱԳԻՐ

ԹԵՄԱ՝ Ճանապարհատրանսպորտային խախտումների հայտնաբերումը մեքենայական ուսուցման կիրառմամբ

Եղիգարյան Ռուբեն Ռոբերտի

Սույն մագիստրական ատենախոսության շրջանակներում կատարվելու է ճանապարհատրանսպորտային խախտումների հայտնաբերման համակարգի նախագծում և իրագործում, ալգորիթմի մշակում և ծրագրավորում։

Աշխատանքում օգտագործվելու է մեքենայական ուսուցման մոդել, որի շնորհիվ համակարգը տեսահոլովակի մեջ փնտրելու է մեքենաներ և փորձելու է հասկանալ, թե արդյոք տվյալ տեսահոլովակում կա՞ ճանապարհատրանսպորտային խախտում, թե՝ ոչ։

Առանցքային բառեր. մեքենայական ուսուցում, տեսահոլովակների զննում, ալգորիթմի մշակում, համակարգի մշակում։

Բովանդակություն

1. Տեսական առնչություններ

1.1 Մեքենայական ուսուցում

Նախքան անցնելը բուն թեմային, ծանոթանանք մեքենայական ուսուցման (Machine Learning) հետ: Արթուր Սամուելն այն նկարագրում է այսպես «մեքենայական ուսուցումը մի տեխնոլոգիա է, որը համակարգիչներին հնարավորություն է տալիս սովորելու, առանց բացահայտ ծրագրավորված լինելու»:

Մեքենայական ուսուցման խնդիրներից են.

- Վերահսկվող ուսուցում (Supervised learning)
- Չվերահսկվող ուսուցում (Unsupervised learning)
  - Մասնավոր դեպք է խորհրդատու համակարգը (Recommender system)
- Ուսուցում ամրապնդմամբ (Reinforcement learning)

Վերահսկվող ուսուցման դեպքում մեքենային տրվում է մուտքային տվյալների հավաքածու և այդ տվյալներին համապատասխան ելքային արժեքները: Այսպիսով այս ուսուցման դեպքում մեքենային հայտնի են ամեն մի մուտքային ինֆորմացիային համապատասխանող ելքային արժեքը կամ արժեքները:

1.1.1 Վերահսկվող ուսուցում

Վերահսկվող ուսուցման (Supervised Learning) խնդիրները դասակարգվում են հետևյալ 2 տիպերի՝ ռեգրսիայի խնդիրներ (Regression problems) և դասակարգման խնդիրներ (Classification problems):

Ռեգրեսիայի խնդրներում փորձում ենք կանխատեսել անընդհատ ֆունկցիայի արժեքներ, ինչը նշանակում է, որ մենք փորձում ենք մուտքային փոփոխականները համապատասխանեցնել ինչ-որ անընդհատ ֆունկցիայի ելքային արժեքներին: Դասակարգման հարցում մենք փոխարենը փորձում ենք կանխատեսել ընդհատ ելքային արժեքներ:

1.1.2 Չվերահսկվող ուսուցում

Չվերահսկվող ուսուցումը (Unsupervised Learning) հնարավորություն է տալիս լուծել այնպիսի խնդիրներ, որոնց ելքային արժեքների մասին կա՛մ քիչ ինֆորմացիա ունենք, կա՛մ ընդհանրապես չգիտենք, թե ինչ տեսքի պետք է լինեն: Մենք կարող ենք ստանալ մի այնպիսի ելքային տվյալի կառուցվածք, որի վրա մուտքային տվյալի ազդեցությունն անգամ չգիտենք: Այդ կառուցվածքը հնարավոր է ստանալ տվյալները համախմբելու արդյունքում՝ հիմնված մուտքային տվյալի փոփոխականների միջև կապերի վրա:

1.1.3 Որոշ նշանակումներ

Կատարենք մի քանի նշանակումներ, որոնք կօգտագործվեն հետագայում: $X_1, X_2, \dots X_n$-ով կնշանակենք մուտքային պարամետրերը, Y-ով՝ ելքայինները: $Input^{(i)}{}_1, Input^{(i)}{}_2, \dots Input^{(i)}{}_n$-ը մուտքային պարամետրերի արժեքներն են (տվյալի հատկություններ), իսկ $Output^{(i)}$-ն՝ ելքային պարամետրի արժեքն է, որտեղ՝ $i=1,2,\dots,m$: Հարմարավետության համար $Input^{(i)}{}_1, Input^{(i)}{}_2, \dots Input^{(i)}{}_n$-ը նշանակենք $x^{(i)}$-ով, իսկ $Output^{(i)}$-ն՝ $y^{(i)}$-ով, n-ը մուտքային պարամետրերի քանակն է: $(x^{(i)}, y^{(i)})$ զույգն կանվանենք ուսուցման օրինակ (training example), իսկ դրանց ցուցակը՝ ուսուցման տվյալներ (training set): Այսինքն m-ը՝ ուսուցման տվյալների քանակն է:

Կարող ենք ասել, որ «Վերահսկվող ուսուցման նպատակն է՝ տրված ուսուցման տվյալների հիման վրա ձևավորել մի այնպիսի $h : X \to Y$ հիպոթեզ ֆունկցիա, որ $h(x)$-ի ելքային արժեքը բավարար մոտ լինի համապատասխան y-ի արժեքին»: Ինչքան $h(x)$-ի արժեքը մոտ լինի համապատասխան y-ի արժեքին, այնքան ավելի ճիշտ արդյունքներ կտա մեր մեքենայական ուսուցման մոդելը: $\theta_0, \theta_1, \dots \theta_n$-ով նշանակենք $h(x)$-ի գործակիցները (որոշ դեպքերում $h(x)$-ը կնշանակենք $h_\theta(x)$): Մեքենայական ուսուցման խնդիրը հենց այդ θ-ների արժեքները գտնելու մեջ է կայանում, քանզի, հետագայում՝ երբ արդեն մեր մոդելը բավարար չափով ուսուցանված կլինի, նրան տրվելու են $X_1, X_2, \dots X_n$ արժեքները և քանզի այն ունի արդեն հաշվարկած $\theta_0, \theta_1, \dots \theta_n$ արժեքները, ընդամենը պետք է հաշվի $h_\theta(x)$-ի արժեքը:

1.2 Ուսուցման տարրեր

1.2.1 Արժեքի ֆունկցիա

h(x)-ի արժեքների ճշտությունը կարելի է գնահատել **արժեքի ֆունկցիայի (Cost Function)** միջոցով: Այն իրենից ներկայացնում է *h(x)-ի* բոլոր ելքային արժեքների և իրական *y-ների* արժեքների միջինացված տարբերություն (Բձ. 1):

$$J(\theta_0, \theta_1, \dots \theta_n) = \frac{1}{2m}\sum_{i=1}^{m}(h(x_i) - y_i)^2 \qquad (1)$$

Այս ֆունկցիան նաև կոչվում է քառակուսային սխալի ֆունկցիա (Squared error function): Քառակուսային միջինը բաժանվել է 2-ի՝ հետագա հաշվարկների հարմարավետության համար, քանի որ դրա միջոցով $(h_\theta(x_i) - y_i)^2$-ի ածանցումից ստացված քառակուսի աստիճանը կվերանա:

Ստացվեց, որ մեր խնդիրը կայանում է *J(θ₀, θ₁, … θₙ)* - ը մինիմիզացնելու մեջ, որի ֆորմալ տեսքը ներկայացված է բանաձև 2-ում

$$\underset{\theta_0, \theta_1, \dots \theta_n}{minimize} J(\theta_0, \theta_1, \dots \theta_n) \qquad (2)$$

1.2.2 Նվազող գրադիենտ

Այսպիսով արդեն պարզաբանվեց, թե ինչ է հիպոթեզ ֆունկցիան և թե ինչպես կարելի է չափել նրա ճշտությունը: Այժմ անհրաժեշտ է որոշել հիպոթեզի պարամետրերը:

Քանզի արժեքի ֆունկցիան հիմնականում իրենից ներկայացնում է բարդ մաթեմատիկական բանաձև, այն դժվար է գծել, կամ գտնել, թե *θ-ի* որ արժեքների դեպքում է այն ընդունում մինիմալ արժեք: Հենց այս խնդիրը լուծելու համար օգտագործվում է նվազող գրադիենտը (Gradient Descent):

Կամայական ֆունկցիայի ածանցյալը ցույց է տալիս տվյալ կետում շոշափողի ուղղությունը, հետևաբար ամեն քայլին շարժվելով այն ուղղությամբ, որն ամենաշատն է նվազեցնում արժեքի ֆունկցիան ի վերջո կհասնենք որևէ մինիմում արժեքի: Յուրաքանչյուր քայլի չափը որոշվում է α պարամետրի միջոցով, որը կոչվում է ուսուցման

գործակից (learning rate): Քայլի ուղղությունը, որոշվում է $J(\theta_0, \theta_1, \dots \theta_n)$-ի մասնակի ածանցյալով: Կախված այն բանից, թե որտեղից ենք սկսում դիտարկել գրաֆիկը, հնարավոր է տարբեր մինիմումների հասնել:

Ընդհանուր դեպքի համար նվազող գրադիենտի ալգորիթմը կլինի. կրկնել հետևյալը մինչև զուգամիտում՝ $\theta_j := \theta_j - \alpha \frac{\delta}{\delta \theta_j} J(\theta_0, \theta_1, \dots \theta_n)$, որտեղ՝ $j = 0,1, \dots n$ ներկայացնում է հատկության հերթական համարը: Այն անվանում են նաև թարմացման կանոն (update rule)*:* Յուրաքանչյուր իտերացիային պետք է միաժամանակ թարմացնել բոլոր $\theta_0, \theta_1, \dots \theta_n$ պարամետրերը:

Պետք է հաշվի առնել, որ կարևոր է α-ի ճիշտ ընտրությունը, քանզի դրանով է պայմանավորված ալգորիթմի զուգամիտման ժամանակը: Եթե ալգորիթմը չի զուգամիտում կամ շատ ժամանակ է պահանջում մինիմումին հասնելու համար ապա α քայլաչափը սխալ է ընտրված: α-ն հաստատուն պահելու դեպքում $\alpha \frac{\delta}{\delta \theta_j} J(\theta_0, \theta_1, \dots \theta_n)$ արտադրյալը ամեն քայլին կնվազի և հասնելով որևէ մինիմումի այն կհավասարվի *0*-ի (իրականում 0-ի չի հավասարվի, այլ կմոտենա ինչ-որ շատ փոքր թվի, որը մեր խնդրի համար համարվում է բավարար) և հետագա քայլերը ոչ մի կերպով չեն ազդի *θ*-ների արժեքների վրա: Հեշտությամբ կարելի է համոզվել, որ, եթե մեր հիպոթեզն ունի գծային տեսք՝

$$h_\theta(x) = \theta_0 + \theta_1 X_1 + \theta_2 X_2 + \dots + \theta_n X_n \qquad (3)$$

ապա թարմացման կանոնի մեջ *J(θ)*-ի արժեքը տեղադրելուց հետո թարմացման կանոնի տեսքը կլինի՝

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) \cdot x_j^{(i)}, \qquad (4)$$

որտեղ՝ j:=0…n: Այստեղ և հետագայում կընդունենք, որ $x_0^{(i)} = 1$, բոլոր *i*-երի համար: Սա արվում է բանաձևերը հարմար ներկայացնելու համար:

1.2.3 Ուսուցման գործակից

Նվազող գրադիենտն իրականացնելուց հետո անհրաժեշտ է հետևել ալգորիթմի աշխատանքին (մոդելի ուսուցման պրոցեսին) և հասկանալ արդյո՞ք այն ճիշտ է աշխատում: Պատկերացում կազմելու համար, թե ինչքան լավ է սովորում մոդելը, անհրաժեշտ է գծել արժեքի ֆունկցիայի՝ *J(θ)-ի,* կախումը *իտերացիաների քանակից:* Եթե ամեն ինչ ճիշտ է աշխատում, ապա ամեն իտերացիայից հետո *J(θ)-ի* արժեքը պետք է նվազի՝ ձգտելով *0-ի:* Հետևաբար, եթե գրաֆիկը աճում է, ապա ինչ որ բան այն չէ: Հիմնականում դրա պատճառը *α-ի* մեծ արժեքն է լինում: Հարկ է նշել՝ ապացուցված է, որ, եթե ուսուցման գործակից (Learning Rate) *α-ն* բավարար չափով փոքր է ընտրված, ապա *J(θ)-ն* նվազում է ամեն իտերացիայի: Սակայն, եթե այն շատ փոքր է ընտրված, ապա *J(θ)-ն* կարող է շատ դանդաղ նվազել:

Կարելի է համարել որ մոդելը բավարար չափով ուսուցանվել է միայն, երբ *J(θ)-ի* փոփոխությունն ինչ-որ իտերացիայից հետո փոքր է որևէ *E* արժեքից: *E-ն* կամայապես ընտրված փոքր թիվ է, օրինակ՝ 10^{-3}: Գործնականում դժվար է ընտրել *E-ի* օպտիմալ արժեք:

1.2.4 Մուտքային տվյալի հատկության մասշտաբավորում

Մենք կարող ենք արագացնել նվազող գրադիենտի աշխատանքը՝ բերելով բոլոր մուտքային պարամետրերը մոտավորապես նույն տիրույթի թվերի: Դա կապված է այն բանի հետ, որ որ *θ-ն* ավելի արագ է հասնում մինիմումին փոքր միջակայքերում և ավելի դանդաղ՝ մեծ միջակայքերում, հետևաբար այն տատանվելով է այն տատանվելով է ձգտում մինիմումին, երբ փոփոխականները շատ անհավասար են: Դա կանխելու համար կարող ենք այնպես փոփոխել հատկությունները, որ նրանք ընկնեն մոտավորապես միևնույն թվային տիրույթ: Իդեալական դեպքում՝ $-1 < x_i < 1$ կամ՝ $-0.5 < x_i < 0.5$: Եթե չկատարվի հատկությունների մասշտաբավորում, ապա որոշ դեպքերում հնարավոր է, որ ալգորիթմը երբեք չզուգամիտի:

Հատկության մասշտաբավորումն (Feature Scaling) ու միջինով նորմալացումը (Mean Normalization) այն երկու մեթոդներն են, որոնք կօգնեն լուծել այդ խնդիրը: Առաջինը

ենթադրում է մուտքային տվյալների բաժանում նրանց մեծագույն և փոքրագույն արժեքների տարբերության վրա: Միջինով նորմալացման դեպքում պետք է մուտքային փոփոխականից հանել մուտքային տվյալների միջին արժեքը, ապա նոր բաժանել մեծագույն և փոքրագույն արժեքների տարբերության վրա: Այս երկու մեթոդների իրականացման համար անհրաժեշտ է փոփոխել մուտքային պարամետրերը՝ համապատասխան ներքևի բանաձևի.

$$x_i := \frac{x_i - \mu_i}{s_i}, \tag{5}$$

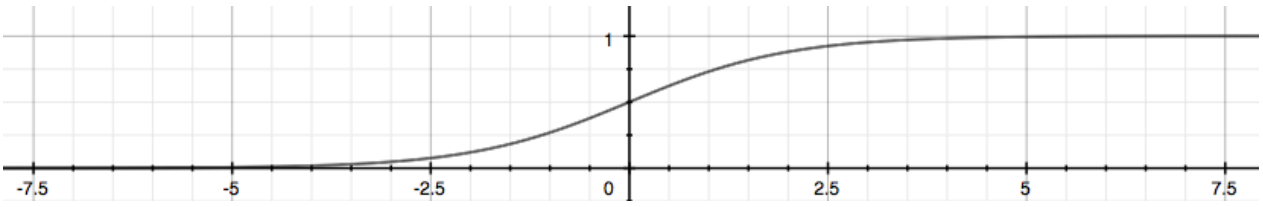
որտեղ s_i-ն i-րդ հատկության մեծագույն և փոքրագույն արժեքների տարբերությունն է, իսկ μ_i-ն՝ այդ հատկության բոլոր արժեքների միջինը:

1.3 Դասակարգում

Երկուական դասակարգման խնդիր (binary classification problem), որտեղ *y*-ը կարող է ընդունել միայն 2 արժեք՝ *0* և *1,* լուծելու համար համար կձևափոխենք $h_\theta(x)$-ն այնպես, որ այն բավարարի $0 \le h_\theta(x) \le 1$ պայմանին: Դա անելու համար կարելի է լոգիստիկ ֆունկցիային (Logistic Function) փոխանցել $\theta^T x$-ը.

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} \tag{6}$$

Այստեղ *g*-ն հենց այն լոգիստիկ ֆունկցիան է, որը կամայական իրական թիվ համապատասխանեցնում է (0, 1) տիրույթի որևէ թվի, ինչը թույլ է տալիս կամայական տիրույթի ելքային արժեքներ ունեցող ֆունկցիան փոխակերպել դասակարգման խնդրին ավելի հարմար ֆունկցիայի: Լոգիստիկ ֆունկցիան նաև անվանում են Սիգմոիդ ֆունկցիա (Sigmoid Function): Այսպիսով $h_\theta(x)$-ը ելքային արժեքի 1 լինելու հավանականությունն է: Նկ. 1-ում պատկերված է այդպիսի ֆունկցիայի մի օրինակ:



Նկ. 1 Սիգմոիդ ֆունկցիայի օրինակ

1.3.1 Լոգիստիկ հիպոթեզի արժեքի ֆունկցիան

Ընդհանուր դեպքում արժեքի ֆունկցիան ունի հետևյալ տեսքը՝

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x_i), y_i), \tag{7}$$

որտեղ *Cost*-ը այն ֆունկցիան է, որը հաշվում է արժեքը *i*-րդ ուսուցման օրինակի համար: Լոգիստիկ ֆունկցիայի համար կարելի է օգտագործել հետևյալ ֆունկցիան՝

$$\begin{cases} Cost(h_\theta(x), y) = -\log\big(h_\theta(x)\big) & \text{երբ՝ } y = 1 \\ Cost(h_\theta(x), y) = -\log\big(1 - h_\theta(x)\big) & \text{երբ՝ } y = 0 \end{cases} \tag{8}$$

Այստեղից երևում է, որ, եթե արժեքի ֆունկցիան գրենք այս ձևով, ապա համոզված կարող ենք ասել, որ *J*-ն ունի ուռուցիկ տեսք լոգիստիկ ռեգրեսիայի համար: Ինչը շատ կարևոր է ավելի արագ ուսուցանվող և ճիշտ արդյունքներ գուշակող մոդել ստեղծելու համար: Այն կարելի է փոխարինել մեկ արտահայտությամբ հետևյալ կերպ՝

$$Cost(h_\theta(x), y) = -y\log\big(h_\theta(x)\big) - (1-y)\log\big(1 - h_\theta(x)\big) \tag{9}$$

Հետևաբար *J*-ն կունենա հետևյալ տեսքը՝

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right) - \left(1-y^{(i)}\right)\log\left(1 - h_\theta\left(x^{(i)}\right)\right)\right] \tag{10}$$

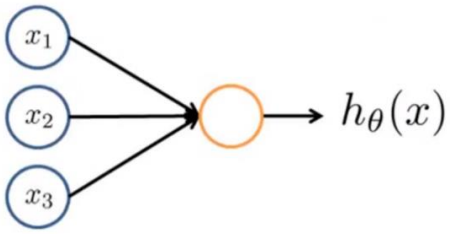
1.4 Նեյրոնային ցանցեր

Հասկանալու համար, թե ինչ անհրաժեշտություն կա ուսումնասիրել այլ ուսուցման ալգորիթմ՝ նեյրոնային ցանցեր (Neural Networks), պատկերացնենք մի դեպք, երբ դասակարգման խնդիր լուծելիս հատկությունները բավարար չեն ճշգրիտ մոդել ուսուցանելու համար, և անհրաժեշտություն է առաջացել ավելացնել նոր՝ քառակուսային, խորանարդային կամ այլ,հատկություններ: Այս դեպքում եթե ավելացնենք բոլոր քառակուսային հատկությունները՝

$$\begin{gathered} x_1^2, x_1x_2, x_1x_3, \dots x_1x_n \\ x_2^2, x_2x_3, \dots x_2x_n \\ \dots \\ x_{n-1}^2, x_{n-1}x_n \\ x_n^2 \end{gathered} \tag{11}$$

ապա կստանանք $\frac{n\cdot(n+1)}{2}+n$ քանակի հատկություն: Այսինքն ստացվում է, որ նոր հատկությունների քանակը նախկինից մոտավորապես քառակուսային $\left(\approx \frac{n^2}{2}\right)$ կախում ունի: Նույն ձևով խորանարդային հատկություններ ավելացնելիս կարելի է համոզվել որ կախումը խորանարդային է: Սա կարող է բերել գերհամապատասխանեցման (overfitting) խնդրին ինչպես նաև բավականաչափ մեծ հաշվողական ռեսուրսներ կպահանջվեն նման մեծ թվով հատկությունների հետ աշխատելու համար:

Մեքենայական ուսուցման մեջ օգտագործվող նեյրոնի մոդելը հնարավորինս մոտ է արված մարդու ուղեղի նեյրոնային կառուցվածքին: Յուրաքանչյուր նեյրոն ստանում է մուտքին որևէ պարամետրեր, կատարում է որոշակի հաշվարկներ, և արդյուների հիման վրա որոշում է թե ինչ ազդանշան ուղարկի հաջորդ նեյրոնին:

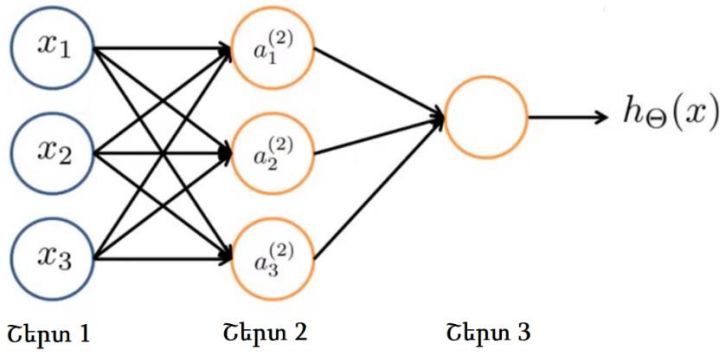


Նկ. 2 Նեյրոնի մոդելի օրինակ

Նկ. 2-ում պատկերված է նեյրոնի պարզեցված մոդելը, որն օգտագործվում է մեքենայական ուսուցման մեջ: x_1, x_2, x_3-ը մուտքային պարամետրերն են, իսկ դեղինով եզրագծվածը նեյրոնի «մարմինն» է: Այստեղ նույնպես կարող ենք ավելացնել $x_0 = 1$ պարամետրը, որը կոչվում է շեղում (bias): Նեյրոնի կատարած հաշվարկների արդյունքը հիպոթեզ ֆունկցիայի արժեքն է,որի բանաձևը լոգիստիկ ռեգրեսիայի բանաձևն է: Նեյրոնի հիպոթեզի ֆունկցիան այլ կերպ անվանում են նաև սիգմոիդ ակտիվացման ֆունկցիա: Նեյրոնի ակտիվացիան դա լոկ այն արժեքն է,որը հաշվարկում է այդ նեյրոնը,այլ կերպ ասած նրա ելքում ստացված արժեքն է: Բնականաբար այդ ֆունկցիան,ինչպես և նախորդ մեր դիտարկած սիգմոիդ ֆունկցիան ունի իր պարամետրերը՝ $\theta_0, \theta_1, \ldots \theta_n$ (մեր օրինակի դեպքում n = 3), որոնց անվանում են կշիռներ (weights):

Նեյրոնային ցանցը, ինչպես բխում է անունից, բազմաթիվ նեյրոններից բաղկացած ցանց է, որոնց ելքերն ու մուտքերը կապված են միմյանց հետ: Նկ. 3-ում պատկերված է նեյրոնային ցանցի մի պարզ օրինակ:

Ցանցը բաժանվում է շերտերի (layers): Առաջին շերտը անվանում են մուտքային շերտ, քանի որ սա այն շերտն է որտեղ ցանցի մուտքին տրվում են հատկությունները: Վերջին շերտը անվանում են ելքային շերտ, այն հաշվարկում է հիպոթեզ ֆունկցիայի վերջնական արժեքը: Առաջին և վերջին շերտերի միջև ընկած բոլոր մնացած շերտերն անվանում են թաքնված շերտեր: Վերջիններս թաքնված են, քանի որ ուսուցման ընթացքում նրանց արժեքներին չենք հետևում: $a_i^{(j)}$-ով նշանակված է j-րդ շերտի i-րդ նեյրոնի ակտիվացիան», իսկ $\theta^{(j)}$-ով կնշանակենք կշիռների այն մատրիցը, որը պարունակում է j-ից (j+1) շերտ անցնելու բոլոր ակտիվացիաների ֆունկցիաների պարամետրերը:



Նկ. 3 Նեյրոնային ցանցի պարզ օրինակ

Ընդհանուր դեպքում (j+1)-րդ շերտի նեյրոնների ակտիվացման ֆունկցիաների տեսքը բերված է ստորև.

$$a_1^{(j+1)} = g(\theta_{10}^{(j)} x_0 + \theta_{11}^{(j)} x_1 + \ldots + \theta_{1n}^{(j)} x_n) \quad (12)$$

$$a_{s_{j+1}}^{(j+1)} = g(\theta_{s_{j+1}0}^{(j)} x_0 + \theta_{s_{j+1}1}^{(j)} x_1 + \ldots + \theta_{s_{j+1}n}^{(j)} x_n) \quad (13)$$

որտեղ n-ը մուտքային պարամետրերի քանակն է, իսկ s_{j+1}-ը՝ (j+1)-րդ շերտում նեյրոնների քանակը: g-ֆունկցիան արդեն պարզաբանվել է 1.3 բաժնում: Նշված ֆունկցիաների օգնությամբ վերջին շերտի արժեքը հաշվելով կարող ենք ստանալ $h_\theta(x)$-ի արժեքը:

2. Գեներատիվ մրցակցող ցանցեր

2.1 Մրցակցող ցանցեր

2.1.1 Մինիմաքս ալգորիթմ

Մինիմաքսը որոշումներ ընդունելու կանոն է, որն օգտագործվում է արհեստական բանականությունության, որոշումների տեսության, խաղերի տեսության, ստատիստիկայի և փիլիսոփայության մեջ, հնարավոր կորուստը (վատագույն դեպքում՝ մաքսիմալ կորուստը) քչացնելու համար։ Սկզբում կանոնները նախատեսված էին երկու խաղացողից բաղկացած զրոյական գումարով խաղի համար, որտեղ մի խաղացողի հաղթանակը բացառում է մյուսի հաղթանակը։ Հետագայում այն զարգացել է և օգտագործվում է ավելի բարդ խաղերում, ինչպես նաև անորոշության պայմաններում որոշումների ընդունման մեջ։

Խաղացողի մաքսիմալ վաստակած միավորը դա այն ամենամեծ թիվն է, որը խաղացողը կարող է հավաքել, առանց իմանալու հակառակորդների քայլերը։ Համապատասխանաբար այդ միավորը այն ամենափոքր թիվն է, որը հակառակորդները կարող են ստիպել խաղացողին հավաքել, երբ գիտեն նրա քայլերը։ Ասվածը մաթեմատիկորեն կարող ենք ներկայացնել հետևյալ կերպ՝

$$\underline{v_i} = \max_{a_i} \min_{a_{-i}} v_i(a_i, a_{-i}) \tag{14}$$

Որտեղ՝

- i-ն հերթական խաղացողի համարն է
- (-i)-ն բոլոր խաղացողներն են՝ բացառությամբ i-րդի
- a_i-ն i-րդ խաղացողի քայլն է
- a_{-i}-ն բոլոր խաղացողների քայլերն են՝ բացառությամբ i-րդի
- v_i i-րդ խաղացողի միավորների հաշվման (արժեքի) ֆունկցիան է

i-րդ խաղացողի մաքսիմալ միավորի հաշվարկը կատարվում է հաշվի առնելով վատագույն տարբերակը՝ նրա ամեն մի հնարավոր քայլի համար ստուգվում է մնացած խաղացողների հնարավոր բոլոր քայլերը և գտնվում է վատագույն կոմբինացիան, որը

խաղացողին կբերի ամենափոքր միավորը: Հետո պետք է հասկանալ, թե, ինչ քայլ պետք է անի i-րդ խաղացողը, որպեսզի համոզված լինի, որ այս ամենափոքր միավորը դա նրա ամենամեծ հնարավոր միավորն է: Այսինքն i-րդ խաղացողն իր հերթական քայլով մաքսիմիզացնում է իր միավորը և մինիմիզացնում է հակառակորդների ազդեցությունը իր միավորի վրա:

Այն խաղը որում հնարավոր է կիրառել վերը նշված մոտեցումը, անվանում են «Մինիմաքս խաղ»:

2.1.2 Գեներատիվ մրցակցող ցանցեր աշխատանքի նկարագրություն

Մրցակցող ցանցերի առաջին հայտնագործողը Գուդֆելոուն էր: Նա առաջինն էր, ով մտածեց հետևյալ հարցի շուրջ՝ «Ի՞նչ կլինի եթե երկու նեյրոնային ցանցի ստիպենք մրցակցել միմյանց դեմ կամայական տիպի տվյալներ գեներացնելու համար»: Ընդ որում մրցակցող ցանցերից առաջինը տվյալների գեներատորն է, իսկ երկրորդը՝ տարբերակիչը: Այստեղ տվյալ ասելով հասկանում ենք ամեն ինչ՝ նկար, երաժշտություն, վիդեո, տեքստ և այլն: Գեներատորը, ինչպես հետևում է իր անվանումից, տվյալներ գեներացնող մոդելն է, իսկ տարբերակիչը իրական տվյալները գեներացվածներից տարբերակող մոդելն է: Հետագա փորձերի արդյունքները բավականին լավ արդյունքներ ցույց տվեցին և այն, ինչը հայտնաբերեց Գուդֆելոուն կոչվեց «գեներատիվ մրցակցող ցանցեր» (ԳՄՑ): ԳՄՑ-ի միջոցով հնարավոր դարձավ ուսուցանել մի այնպիսի մոդել, որն ունակ է ստեղծել ուսուցման ընթացքում իրեն տրված տիպի տվյալներ: Օրինակ՝ կենդանիների կամ մեքենաների նկարներ: Վերջին տարիներին խորը ուսուցումը (Deep Learning) բավականին առաջընտաց է ապրել: Դրա միջոցով հնարավոր է ստեղծել մոդելներ, որոնք ունակ են ճանաչել տարբեր տեսակի առարկաներ նկարների մեջ, սակայն հնարավոր չէ ստեղծել մի այնպիսի մոդել որը նկարներ կգեներացնի նույնքան լավ, ինչքան ԳՄՑ-երը:

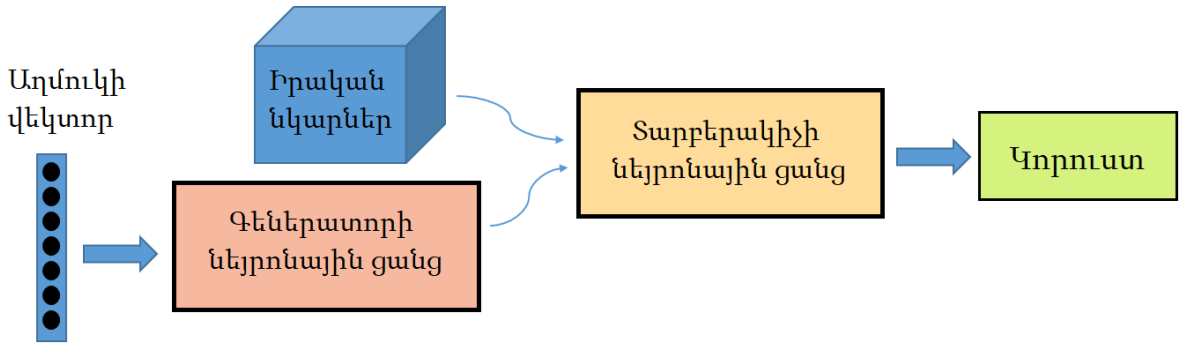
ԳՄՑ-ի միջոցով կարելի է ստանալ նկարների գեներատիվ հզոր մոդելներ, սակայն ստացված մոդելներն ունակ չեն գեներացնել կամայական տիպի տվյալներ: Այդ մոդելները գեներացնում են միայն այնպիսի տվյալներ, ինչի վրա որ կատարվել է ուսուցումը: Այսպիսով, եթե ԳՄՑ-ի ուսուցման ժամանակ տրվել են միայն շան նկարներ,

ապա գեներատիվ մոդելը կսովորի գեներացնել միայն շան նկարներ, և ունակ չի լինի գեներացնել բոլորովին այլ տիպի կենդանու նկարներ: Իսկ եթե մուտքային տվյալները բավականին տարբեր բնույթի լինեն, ապա հնարավոր է որ գեներատորի ուսուցումը անհաջող լինի և այն վերջիվերջո ունակ չլինի գեներացնել որևէ իմաստ արտահայտող տվյալներ:

Սույն աշխատությունում ուսուցման ժամանակ G գեներատորի մուտքին տրվելու է որևէ z բաշխումից աղմուկ՝ $p_z(z)$, որից G-ն ստանալու է նոր նկար, այդ ֆունկցիան նշանակենք՝ $G(z;\theta_g)$: Սահմանենք ևս մեկ ֆունկցիա տարբերակիչի համար՝ $D(x;\theta_d)$, որի ելքը մի սկալյար մեծություն է, որն արտահայտում է նրա մուտքին տրված x նկարի իրական լինելու հավանականությունը: Այսինքն նրա ելքը կլինի 0, եթե մուտքին տրված նկարը գեներացված է, և 1՝ հակառակ դեպքում: Ստացվում է, որ D-ն մեզ մոտ երկուական դասակարգիչ է: Ուսուցման ընթացքում մենք փորձում ենք մեծացնել D-ի ճշտությունը, նրա մուտքին տալով իրական և գեներատորի գեներացրած նկարներ: Միևնույն ժամանակ՝ զուգահեռաբար, մենք ուսուցանում ենք G-ն ստիպելով նրան փոքրացնել D-ի ճշտությունը: Այլ կերպ ասած այս երկու մոդելները մրցակցում են միմյանց հետ և խաղում են հետևյալ մինիմաքս խաղը $V(G,D)$ արժեքի ֆունկցիայով՝

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \qquad (15)$$

ԳՄՑ ուսուցանելիս պետք է միշտ հիշել, որ մենք ուսուցանում ենք գեներատորը և տարբերակիչը զուգահեռաբար: Այսինքն ուսուցման մեկ քայլի ընթացքում նեյրոնային ցանցի կշիռները թարմացնում է թե՛ գեներացնող և թե՛ տարբերակող մոդելը: Նկ. 4-ում պատկերված են ուսուցմանը մասնակցող նեյրոնային ցանցերը և նրանց միջև կապերն ու մուտքերը:



Նկ. 4 Գեներատիվ մրցակցող ցանցերի աշխատանքի սխեմա

2.2 Խորը փաթույթային գեներատիվ մրցակցող ցանցեր

Ներկա պահին գոյություն ունեն ԳՄՑ-երի տասնյակ տարբեր տեսակներ: Խորը փաթույթային գեներատիվ մրցակցող ցանցերը (ԽՓԳՄՑ) դրանցից մեկն է: ԽՓԳՄՑ-ն նախատեսված է նկարների գեներացիայի համար: Երկար տարիներ ուշադրության կենտրոնում են եղել վերահսկվող փաթույթային նեյրոնային ցանցերը, մինչդեռ ԽՓԳՄՑ-ն լավ օրինակ է չվերահսկվող փաթույթային նեյրոնային ցանցերի ոչ պակաս արդյունավետության: Ինչպես երևում է անվանումից, ԽՓԳՄՑ-երի հիմքում ընկած է փաթույթային ցանցերի գաղափարը, որոշակի փոփոխություններով, որոնք առաջ են քաշվել վերջերս: Դրանք են՝

1. Նեյրոնային ցանցի բոլոր ոչ-փաթույթային՝ միավորման (pooling), շերտերը փոխարինել փաթույթայինով, ինչը հնարավորություն կտա ցանցին սովորել մշակել սեփական downsampling-ը: Այս աշխատությունում այս մոտեցումը օգտագործվել է նաև գեներատորի մոդելավորման համար, ինչը թույլ է տալիս գեներատորին մշակել իր սեփական upsampling-ը:
2. Վերացնել ամբողջությամբ միացված շերտերը փաթույթային շերտերից առաջ:
3. Անհրաժեշտ է կիրառել խմբային նորմալիզացում (Batch Normalization), ինչը կկայունացնի ուսուցումը: Այն կնորմալիզացնի ամեն նեյրոնի մուտքը՝ բերելով միջին արժեքը զրոյի:
4. Տարբերակիչի բոլոր շերտերի համար օգտագործել բացվածքով ReLU (Leaky ReLU) ֆունկցիան ReLU-ի փոխարեն:

2.2.1 Մրցակցող մոդելներ

Նրցակցող մոդելների զույգն է՝ գեներատոր-տարբերակիչ: Այս 2 մոդելների ուսուցման ժամանակ մենք չենք թարմացնում տարբերակիչի կշիռները, քանի որ այս դեպքում մեր խնդիրը գեներատորի որակը լավացնելն է: Տարբերակիչը թարմացնում է իր կշիռները ուսուցման այլ քայլերի ընթացքում:

Ուսուցումը բաղկացած է հետևյալ քայլերից՝

1. Տարբերակիչի ուսուցում
2. Գեներատոր-Տարբերակիչ մրցակցող մոդելների ուսուցում

Ընդ որում առաջին քայլի ընթացքում թարմացվում են տարբերակիչի կշիռները, իսկ վերջին քայլն ուղղված է գեներատորի որակի լավացմանը, այսինքն միայն գեներատորի կշիռներն են թարմացվում:

Առաջին քայլի ընթացքում տարբերակիչի մուտքին տրվում են ինչպես գեներատորի գեներացրած նկարները, այնպես էլ իրական նկարներ:

2.3 Python լեզուն

Python-ը բարձր կարգի, ինտերպրիտացվող, դինամիկ, սակայն ուժեղ տիպիզացված (Dynamically typed, Strong typed) լեզու է: Նշված հատկությունները Python-ին դարձնում են մեքենայական ուսուցման խնդիրների լուծման համար բավական հարմար լեզու: Բացի նշված հատկություններից, Python լեզվի բարձր տարածվածությունը տվյալագիտության ոլորտում կապված է նաև այնպիսի գործոնների հետ ինչպիսիք են՝ լեզվի պարզ սինտաքս (syntax), բազմաքանակ և զարգացած գրադարանների և հավելումների (plug-in) առկայություն, լավ դոկումենտացիա, մեծ և զարգացած համայնք (community):

Բարձր կարգի լեզվի առավելությունը կայանում է նրանում, որ այն անկախ է պլատֆորմից, ապարատային ապահովումից (hardware) և նրա ճարտարապետությունից: Ինչպես նաև այն ավելի հեշտ է սովորել քան ցածր մակարդակի լեզուները, ինչպես նաև

այն շատ ավելի հեշտ է կարգաբերել (debug)։ Python-ը լինելով բարձր կարգի լեզու, բնականաբար ունի նշված առավելությունները։

Լեզվի պարզ սինտաքսն օգնում է Python-ի սկսնակ ծրագրավորողներին կարճ ժամանակում տիրապետել լեզվին, ինչն արագացնում է կամայական նախագծի ծրագրավորման գործընթացը։ Ինչպես նաև այն, քիչ թե շատ, հեշտացնում է անծանոթ կոդի կարդալն ու հասկանալը։

Դինամիկ տիպիզացված լեզուն շատ հարմար է օգտագործման համար։ Օրինակ հարկ եղած դեպքում կարելի է զանգվածի մեջ պահել տարբեր տիպի օբյեկտներ, ինչը կարող է պետք գալ արագորեն ինչ-որ բան թեստավորելու համար։ Սակայն պետք է զգույշ լինել կատարման ընթացքում բացառիկ իրավիճակներից (Runtime Exceptions) խուսափելու համար։

Վերջին 2 հատկությունների առավելություններն ավելի լավ պատկերացնելու համար դիտարկենք հետևյալ օրինակը՝

$$\text{arr} = [1, 20, -334, 44.44, -555.50005, \text{“str\_val\_1”}, \text{“str\_val\_2”}] \qquad (16)$$

Այստեղ, arr զանգվածի առաջին երեք էլեմենտները ամբողջ թվեր են, հաջորդ երկուսը՝ կոտորակային, իսկ վերջին երկուսն ընդհանրապես թվեր չեն, այլ տողային օբյեկտներ։ Օգտագործելով «+» օպերատորը, կարելի է իրար միացնել 2 կամ ավելի զանգվածներ և ստեղծել նոր զանգված։ Ինչպես նաև կարելի է զանգվածի սկզբից կամ վերջից հեռացնել որոշ էլեմենտներ ընհամենը մեկ տողով։ Օրինակ, քիչ 16 արտահայտությունում օգտագործված arr զանգվածի վերջին 2 էլեմենտները հեռացնելու համար կարող ենք կատարել հետևյալը՝

$$\text{arr\_no\_str} = \text{arr}[:-2] \qquad (17)$$

որից հետո, arr_no_str-ի արդյունքը կլինի՝

$$[1, 20, -334, 44.44, -555.50005] \qquad (18)$$

Python-ի սինտաքսային առանձնահատկություններից կարևոր է նշել նաև զանգվածների ձևափոխությունը։ Օրինակ, 19 արտահայտությունով կարելի է ստանալ, նոր զանգված, որը կպարունակի 16 արտահայտությունում նշված arr զանգվածի էլեմենտների տիպերը,

իսկ 25 արտահայությունով՝ մի զանգված, որը կպարունակի այդ նույն arr զանգվածի բոլոր այն աբողջ թվերի ½ մասը, որոնք բաժանվում են 2-ի։

```
arr_types = [type(x) for x in arr]                                    (19)
```

```
[x/2 for x in arr if isinstance(x, int) and x % 2 == 0]               (20)
```

Համապատասխանաբար 19 և 25 արտահայտությունների արդյունքը կլինեն 21 և 22 արտահայտությունները։

```
[<class 'int'>, <class 'int'>, <class 'int'>, <class 'float'>,
<class 'float'>, <class 'str'>, <class 'str'>]                        (21)
```

```
[10.0, -167.0]                                                        (22)
```

Գրադարանների ու framework-երի լայն ընտրություն. Մեքենայական ուսուցման ալգորիթմների իմպլեմենտացիան բարդ է և շատ ժամանակ կարող է պահանջել։ Լավ թեստավորված ու կազմակերպված միջավայրի հասանելիությունը ծրագրավորողների համար կենսական նշանակություն ունի։

Որպեսզի կրճատվի ծրագրավորման ժամանակը, կարելի է օգտագործել Python-ի framework-ներին ու գրադարաններին։ Ծրագրային գրադարանը նախապես գրված կոդ է, որն օգտագործվում է ծրագրավորողների կողմից հայտնի ծրագրավորման խնդիրներ լուծելու։ Python-ն ունի մեքենայական ուսուցման գրադարանների լայն բազմություն։ Դրանցից մի քանիսը նշված են ներքևում՝

- Keras, TensorFlow և Scikit-learn - մեքենայական ուսուցման համար,
- NumPy, SciPy - բարձր արագագործությամբ գիտական հաշվարկների ու տվյալների վերլուծության համար,
- Pandas - տվյալների վերլուծության և ձևափոխության համար,
- Seaborn - տվյալների վիզուալիզացիայի համար։

Scikit-learn-ը տրամադրում է բազմազան կլասիֆիկացիայի, րեգրեսիայի և կլաստերիզացիայի ալգորիթմներ՝ ներառյալ random forests, gradient boosting, k-means, և DBSCAN ալգորիթմները։ Այն նախատեսված է աշխատելու Python-ի թվային և գիտական գրադարանների հետ՝ NumPy և SciPy ։

Պլատֆորմից անկախություն. խոսքը ծրագրավորման լեզվի, framework-ի, գրադարանի մասին է, որը հնարավորություն է տալիս ծրագրավորողին աշխատել մեկ մեքենայի վրա, բայց օգտագործել ստեղծված ծրագիրն այլ մեքենաների վրա առանց ավելորդ փոփոխությունների: Python-ի առանձնահատկություններից մեկը պլատֆորմից, ապարատային ապահովումից (hardware) և մեքենայի ճարտարապետությունից անկախ լինելն է: Python-ով կոդը կարող է օգտագործվել ստեղծելու կատարողական ֆայլեր բոլոր տարածված օպերացիոն համակարգերի համար:

Մեքենայական ուսուցման մոդելներ ուսուցանելու համար ծրագրավորողները հաճախ օգտվում են այնպիսի ծառայություններից, ինչպիսիք են՝ Google-ը կամ Amazon-ը: Այնուամենայնիվ կան շատ կազմակերպություններ ու անհատներ, որ օգտագործում են իրենց հզոր GPU-ով մեքենաները, և այն փաստը, որ Python-ն անկախ է պլատֆորմից, դարձնում է այդ ամենը շատ ավելի էժան և հեշտ:

Մեծ և զարգացած համայնք (community). Stack Overflow-ի կողմից 2018թ.-ին անցկացված հարցման համաձայն Python-ը 10 ամենատարածված ծրագրավորման լեզուների ցանկում է: Իսկ ներքևում բերված գրաֆիկից ակնհայտ է, որ Python-ը մարդկանց կողմից Google-ով ամենահաճախ փնտրված լեզուն է:

Օնլայն պահեստները (repositories) պարունակում են ավելի քան 140,000 Python-ով ծրագրավորված նախագծեր: Այդ ծրագրային ապահովումները հնարավորություն է տալիս ծրագրավորողներին առանձնացնել փաթեթերները տվյալների մեծ բազմություններում: