LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

TRANSPORTLĪDZEKĻU APDROŠINĀŠANAS ATLĪDZĪBU PROGNOZĒŠANA IZMANTOJOT DZIĻOS NEIRONU TĪKLUS

MAĢISTRA DARBS

Autors: **Imants Petrovs**
Stud. apl. nr. IP16033
Darba vadītājs: Dr. dat., Prof. Guntis Bārzdiņš

RĪGA 2019

ANOTĀCIJA

Lai apdrošināšanas produkti darbotos veiksmīgi, viens no būtiskākajiem faktoriem ir apdrošināšana risku izvērtēšana. Viena no risku izvērtēšanas sastāvdaļām ir iespējamo apdrošināšanas atlīdzības apmēra prognozēšana, balstoties gan uz apdrošinātā objekta raksturlielumiem, gan klienta parametriem un vēsturi.

Šajā darbā tika apskatītas dziļo neironu tīklu tehnoloģiju pielietojums apdrošināšanas atlīdzību summu novērtēšanā un balstoties uz transportlīdzekļu apdrošināšanas atlīdzību datiem, tika izveidots un apmācīts dziļo neironu tīkla modelis. Tika izveidots risinājums, ar kura palīdzību var prognozēt apdrošināšanas atlīdzību rezervju lielumu nākamajiem periodiem, izmantojot klienta un apdrošināšanas objekta datus. Transportlīdzekļu apdrošināšana tika izvēlēta tāpēc, ka tai ir pieejama pietiekami liela datu kopa.

Darba gaitā tika veikta datu analīze un priekšapstrāde, pazīmju vektoru izveide, modeļa trenēšana un iegūto rezultātu analīze. Pētot iegūtos rezultātus, tika konstatēts, ka vairumā gadījumu zaudējumu funkcija dilst gan izmantojot treniņa, gan testa datu kopas, bet iegūtās prognozes konverģē uz datu kopas vidējām vērtībām, kas norāda, ka datu kopas apmērs ir nepietiekams.

Šī darba rezultāti varēs tik izmantoti kā viena no sastāvdaļām tālākai apdrošināšanas atlīdzību rezervju noteikšanai un atlīdzības lieluma prognozēšanai.

Atslēgvārdi: apdrošināšana, zaudējumu atlīdzības, dziļā mašīnmācīšanās, tensorflow, regresija, klasifikācija

ABSTRACT

The theme of master thesis: Motor insurance claims prediction using deep neural networks

For insurance products to work successfully, one of the most important factors is the insurance risk assessment. One of the components of risk assessment is forecasting the amount of possible insurance indemnity based on both the characteristics of the insured object and the customer's parameters and history.

In a Master thesis (MA thesis), the use of deep neural network technologies in the estimation of insurance claims amounts was analyzed and a deep neural network model was developed and trained based on vehicle insurance data. A solution was created to predict the size of insurance claims reserves for future periods using customer and insurance object data. Vehicle insurance was chosen because it has enough data available.

In the course of the MA thesis, data analysis and preprocessing, creation of feature vectors, model training and analysis of the obtained results were performed. By the results obtained in research, it was found that in most cases the loss function decreases both through training and test data sets, but the resulting predictions converge to the average values in the dataset, indicating that the size of the dataset is insufficient.

The results of this MA thesis will be used as one of the components for further determination of insurance claims reserves and forecasting the amount of remuneration.

Keywords: insurance, loss compensation, deep machine learning, tensorflow, regression, classification

AUTOREFERĀTS

Lai apdrošināšanas produkti darbotos veiksmīgi, viens no būtiskākajiem faktoriem ir apdrošināšana risku izvērtēšana. Viena no risku izvērtēšanas sastāvdaļām ir iespējamo apdrošināšanas atlīdzības apmēra prognozēšana, balstoties gan uz apdrošinātā objekta raksturlielumiem, gan klienta parametriem un vēsturi.

Šajā darbā tika apskatītas dziļo neironu tīklu tehnoloģiju pielietojums apdrošināšanas atlīdzību summu novērtēšanā un balstoties uz transportlīdzekļu apdrošināšanas atlīdzību datiem, tika izveidots un apmācīts dziļo neironu tīkla modelis. Tika izveidots risinājums, ar kura palīdzību var prognozēt apdrošināšanas atlīdzību rezervju lielumu nākamajiem periodiem, izmantojot klienta un apdrošināšanas objekta datus. Transportlīdzekļu apdrošināšana tika izvēlēta tāpēc, ka tai ir pieejama pietiekami liela datu kopa.

Darba gaitā tika veikta datu analīze un priekšapstrāde, pazīmju vektoru izveide, modeļa trenēšana un iegūto rezultātu analīze. Pētot iegūtos rezultātus, tika konstatēts, ka vairumā gadījumu zaudējumu funkcija dilst gan izmantojot treniņa, gan testa datu kopas, bet iegūtās prognozes konverģē uz datu kopas vidējām vērtībām, kas norāda, ka datu kopas apmērs ir nepietiekams.

Šī darba rezultāti varēs tik izmantoti kā viena no sastāvdaļām tālākai apdrošināšanas atlīdzību rezervju noteikšanai un atlīdzības lieluma prognozēšanai.

Atslēgvārdi: apdrošināšana, zaudējumu atlīdzības, dziļā mašīnmācīšanās, tensorflow, regresija, klasifikācija

SATURS

APZĪMĒJUMU SARAKSTS

AI – mākslīgais intelekts (*Artificial Intelligence*)

API – lietojumprogrammu saskarne (*Application Programming Interface*)

IDE – integrētā izstrādes vide (*Integrated Development Environment*)

IoT – lietu internets (*Internet of Things*)

CPU – centrālais procesors (*Central processing unit*)

GPU – grafiskais procesors (*Graphics processing unit*)

DNN – dziļie neironu tīkli (*Deep Neural Networks*)

RNN – rekurentie neironu tīkli (*Recurrent Neural Networks*)

ER – Entītiju relācijas

SQL - strukturēta vaicājumu valoda

LTAB - Latvijas Transportlīdzekļu apdrošināšanas birojs

OCTA – Sauszemes transportlīdzekļa īpašnieka civiltiesiskās atbildības obligātā apdrošināšana

KASKO - Brīvprātīga sauszemes transportlīdzekļu apdrošināšana

CSNg – ceļu satiksmes negadījums

BM – apdrošināšanas bonus-malus klase/sistēma

ATVK - administratīvo teritoriju un teritoriālo vienību klasifikators

KPI – izpildes pamat rādītāji (*Key Performance Indicator*)

GLM – vispārinātais lineārais modelis (G*eneral Linear Model*)

IBNR – pieteikto bet neizmaksāto atlīdzību rezerve (*Incurred But Not Reported Reserve*)

IEVADS

Šobrīd dažādas aktuālas mūsdienu tehnoloģijas sāk būtiski mainīt apdrošināšanas nozari. Tādas tehnoloģijas kā mākoņskaitļošana, lietu internets (IoT), telemātikas sistēmas, globālās pozicionēšanas sistēmas (GPS), mobilie tālruņi, digitālās platformas, droni, "blockchain", mākslīgais intelekts piedāvā jaunus veidus un iespējas kā vadīt un novērtēt apdrošināmos riskus, sadarboties ar klientiem, samazināt izdevumus un palielināt efektivitāti. Šīs tehnoloģijas paver arī iespēju radīt jaunus apdrošināšanas produktus, servisus un biznesa modeļus. Pēc Acenture "Technology Vision 2017"[1] pētījuma datiem, 87% respondentu uzskata, ka tehnoloģiju attīstība notiek ne vairs lineārā, bet gan eksponenciālā veidā.

Apdrošināšanas industrijas biznesa modeļa pamātā ir riski, bet datu tehnoloģijas būtiski maina šo risku izvērtēšanas metodes, jo ir pieejami arvien jauni veidi kā iegūt, apkopot un analizēt datus. Pēc "International Data Corporation" [1] pētījuma , ik pēc diviem gadiem, datu apjoms pasaulē dubultojas.

Jau šobrīd eksistē neskaitāmi datu avoti, kas uzkrāj pakalpojumu lietotāju datus reālā laikā, tādā veidā ļaujot arvien labāk aprēķināt un pārvaldīt riskus. Apdrošinātāji sāk izmantot šo pieaugošo datu apjomu, lai samazinātu nezināmo skaitu savos riska modeļos, tādā veidā samazinot riskus un zaudējumus, personalizējot prēmiju aprēķinus un piedāvājot jaunus produktus un pakalpojumus.

Lietu internets ir labs piemērs kā jaunie datu avoti nodrošina iespēju sākt apdrošināšanas nozares transformāciju. Lietu internet ir visdažādākās ierīces, kas ir saslēgtas vienotā tīklā un šie mezgli ļauj uzraudzīt, uzkrāt un dalīties ar datiem internetā. Šo iekārtu skaits nemitīgi pieaug un to skaits 2020. gadā tiek prognozēts vairāk kā 20 miljardi. Šīs tīklā saslēgtās iekārtas ir gan vied ierīces, gan māju un transportlīdzekļu drošības sistēmas, gan virtuves iekārtas, gan kameras, gan sensori (piem. termostati, ūdens noplūdes, u.c.) un daudzi citi. Būtisks izaicinājums ir efektīva datu apjoma izmantošana praktiski lietojamos risinājumos un platformās.

Arvien pieaugošais viedtālruņu skaits kā arī lietotāju pakalpojumu iegādes paradumu maiņa dod iespēju ieviest jaunus risinājumus, kas nodrošina labāku lietošanas pieredzi, informācijas pieeju un pielāgojamību kā tradicionālie apdrošināšanas pakalpojumu sniegšanas kanāli, Pēc Munich RE prognozēm **Error! Reference source not found.**, 2020. gadā tiešsaistes p ārdošanas apjoms pieaugs par 25%. Līdz ar dalīšanās ekonomikas pieaugumu un šīs ekonomikas pārstāvju (tādu kā Airbnb, Uber, u.c.) popularitātes kāpumu, pieaug arī nepieciešamība ieviest "pēc pieprasījuma" apdrošināšanas produktus un platformas. Piemērs ir

apdrošināšanas kompānija “Trōv” (https://trov.com/), kas nodrošina kustamā īpašuma apdrošināšanu uz ļoti īsiem termiņiem.

Ja salīdzinām tehnoloģijas, kas eksistēja kaut vai dažus gadus iepriekš un to, kas ir tagad, var secināt, ka tās kļūst arvien interaktīvākas. Tādas tehnoloģijas kā skārienjūtīgie ekrāni, “jauktā” realitāte, dabiskās valodas apstrādes procesi, kļūst arvien cilvēkam draudzīgāki. Izmantojot konteksta analīzi, attēlu atpazīšanu un dziļās mašīnmācīšanās algoritmus, šīm tehnoloģijām ir paradījusies iespēja mācīties, kas ļauj tās gan personificēt, gan pielāgot dažādu industriju vajadzībām.

Vēsturiski apdrošināšanas industrija nav bijusi līdere ar lietotāja pieredzi saistīto inovāciju jomā. Šobrīd, līdz ar AI tehnoloģiju ienākšanu, šī konservatīvā pieeja būtiski mainās. Piemēram AI var tikt pielietots zaudējumu regulēšanas procesā. Ja esošajā apdrošināšanas zaudējumu regulēšanas procesā ir iesaistīti daudz darbinieku, tad izmantojot AI un citas tehnoloģijas var automatizēt gan zaudējumu pieteikšanas, gan zaudējumu apmēra noteikšanas, gan komunikācijas procesus. Otrs būtisks aspekts zaudējumu regulēšanas procesā ir krāpniecisku darbību identificēšana. Šobrīd krāpnieciskās darbības, apdrošināšanas industrijai, izmaksā vairāk kā 40 miljardus ASV dolārus gadā. AI algoritmi var identificēt datu modeļus un identificēt krāpnieciskās darbības. Trešais virziens, kur piedalās AI ir: aktuār aprēķini, precīzi risku aprēķini un apdrošināšanas piedāvājumu personifikācija.

1. TRANSPORTLĪDZEKĻU APDROŠINĀŠANAS ATLĪDZĪBU REGULĒŠANA

Lai gan apdrošināšanas nozare ir diezgan konservatīva, dziļo neironu tīklu pielietojums apdrošināšanas industrijā varētu būt ļoti plašs. Viena pielietojumu loks ir līdzīgs citām biznesa nozarēm, tas ir dokumentu atpazīšana, klasifikācija un digitalizācija, virtuālo asistentu pielietojumi gan klientu apkalpošanā, gan komunicējot ar uzņēmuma sadarbības partneriem un iekšējiem darbiniekiem, procesu automatizācija un automātisko lēmumu pieņemšanas sistēmu ieviešana. Otra pielietojumu sfēra ir tieši saistīta ar apdrošināšanas sabiedrību pamatdarbību. Tas ir, apdrošināšanas risku izvērtēšana, lai noteiktu pēc iespējas precīzāku un uzņēmumam peļņu nesošu apdrošināšanas prēmiju, zaudējumu pieteikumu apstrādes procesa automatizācija, apdrošināšanas krāpniecības draudu prognozēšana un novēršana, apdrošināšanas risku pārvaldība.

Dziļās mašīnmācīšanās izmantošanas pielietojums ir iespējams, sākot no atlīdzību pieteikuma reģistrēšanas līdz pat pilnīgai atlīdzību lietas noregulēšanai. Vairākas apdrošināšanas kompānijas ir sākušas atlīdzību regulēšanas procesu automatizāciju, tādejādi uzlabojot klientu apmierinātības līmeni un samazinot atlīdzību lietu noregulēšanas laiku. Mašīnmācīšanās prognozēšanas modeļi ļauj optimizēt atlīdzību izmaksas, kā arī arvien precīzāk noteikt atlīdzību rezervju lielumu, kas ļauj samazināt uzņēmuma rezervēto (iesaldēto) finanšu līdzekļu apjomu.

1.1. Zaudējumu pieteikumu apstrādes process

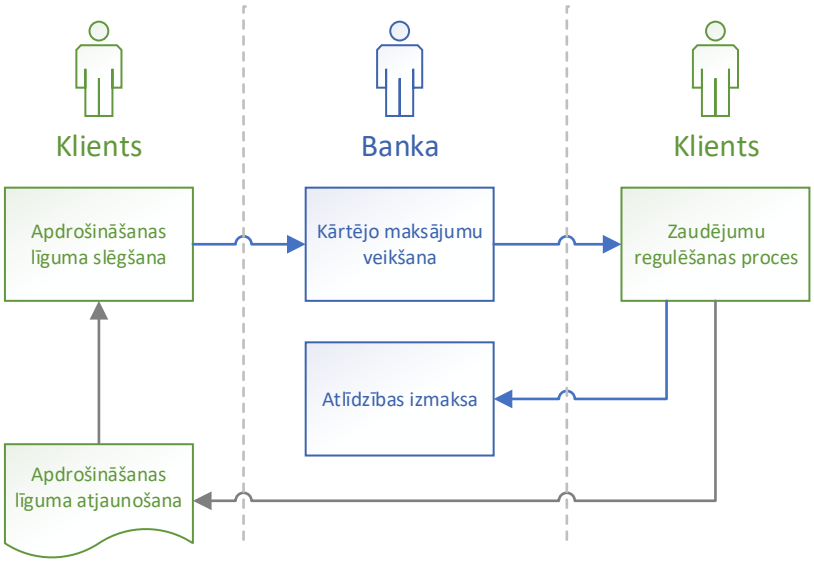
Šajā darbā es apskatīšu divus apdrošināšanas veidus, kur apdrošinātais objekts ir transportlīdzeklis veidi.

KASKO, kas ir brīvprātīgā transportlīdzekļu apdrošināšana pret tādiem riskiem kā zādzība, transportlīdzekļa bojājumi vai bojāeja, vējstiklu bojājumi, transportlīdzekļu aizvietošana, u.c. Dažādām apdrošināšanas kompānijām, apdrošināšanas līgumā iekļautie riski var atšķirties. Visi iespējamie riski un nosacījumi ir atrunāti, katras apdrošināšanas sabiedrības apdrošināšanas veida noteikumos.

OCTA ir obligātā civiltiesiskā transportlīdzekļu apdrošināšana, kas nozīmē, ka šajā apdrošināšanas veidā, tiks atlīdzināti zaudējumi otrajai pusei par polises īpašniek nodarīto kaitējumi. Arī šajā apdrošināšanas veidā apdrošinātais objekts ir transportlīdzeklis. Apdrošinātie riski ir noteikti likumā **Error! Reference source not found.** un tie ietver gan t ransportlīdzekļu bojājumu un bojāejas riskus, gan personu veselībai nodarītā kaitējuma risku, u.c.

1.1.1. Apdrošināšanas gadījumu pieteikšana

Apdrošināšanas sabiedrības izstrādā arvien jaunus veidus kā pieteikt atlīdzību pieteikumus. Šo pasākumu mērķis ir savākt pēc iespējas precīzākus un kvalitatīvākus datus, lai nodrošinātu pēc iespējas kvalitatīvāku klientu apkalpošanas procesu un nodrošinātu atlīdzību lieluma novērtēšanu, kā arī nodrošinātu apdrošināšanas risku novērtēšanu nākamajiem apdrošināšanas līgumiem, lai noteiktu atbilstošu apdrošināšanas prēmiju. Šo datu apkopošana un aprēķinu precizitāte ir svarīga, lai no vienas puses neaizbiedētu klientus ar lielām apdrošināšanas iemaksām, bet no otras puses, ja aprēķinātā apdrošināšanas prēmija ir par mazu, lai neradītu zaudējumus uzņēmumam. Vācot informāciju par apdrošināšanas gadījumiem, apdrošināšanas uzņēmumiem ir jāmeklē kompromiss starp datu detalizāciju un nepieciešamo laiku, kas vajadzīgs klientiem, lai sniegtu šo informāciju. Atšķirībā no banku biznesa, kur klients atrodas nepārtrauktā mijiedarbībā ar banku, veicot maksājumus, apdrošināšanas procesā klients griežas pie apdrošinātāja, galvenokārt, tikai slēdzot līgumu un pieprasot zaudējumu atlīdzību.

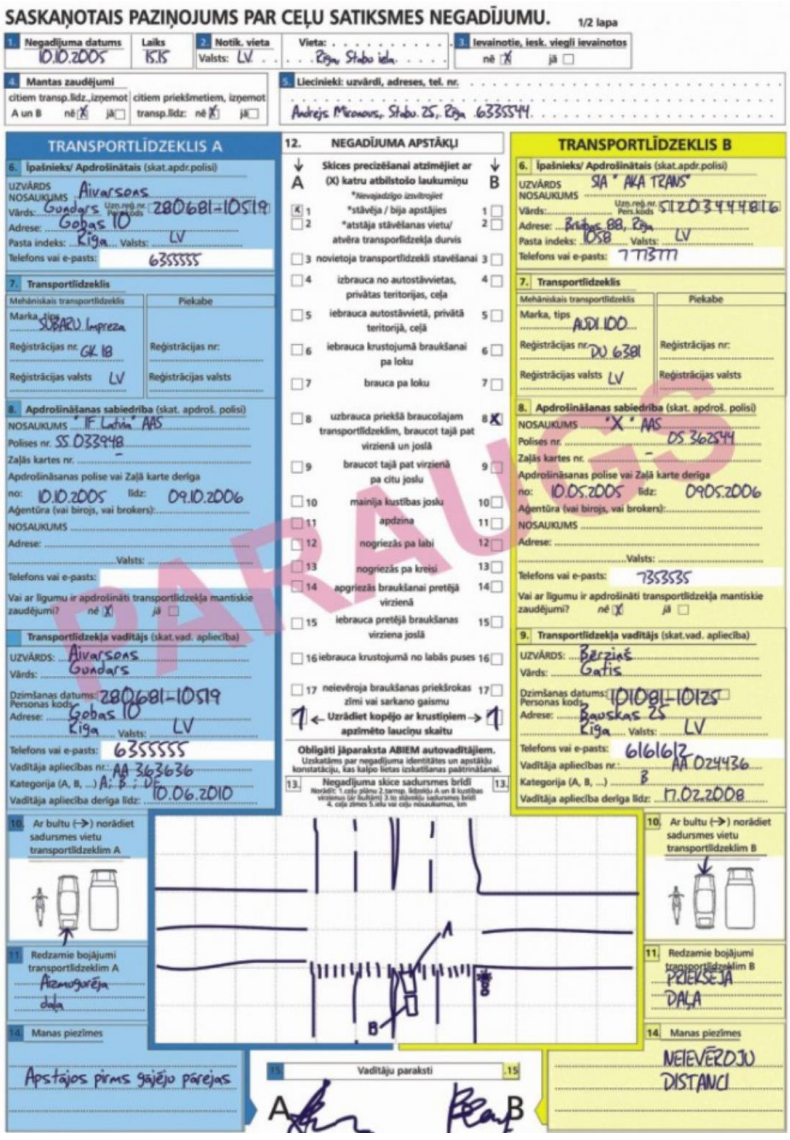


1.1. att. **Apdrošināšanas process no klienta skatu punkta**

Līdz ar to ir apdrošinātājiem ir svarīgi, tajos procesa posmos, kuros klients ir pieejams, savākt nepieciešamo informāciju, iegūt atgriezenisko saiti un vienlaicīgi nodrošinātu augstu klientu apkalpošanas kvalitāti. Lai sasniegtu šo mērķi, apdrošināšanas uzņēmumiem ir jāveido procesuāli un tehnoloģiski risinājumi, gan veicot pieejamo datu analīzi un apstrādi, gan veidojot saskarnes ar trešo pušu piedāvātajiem servisiem, piemēram, LTAB un CSDD datu bāzes, Lursoft uzņēmumu reģistrs, u.c. Līdz ar to apdrošināšanas gadījuma pieteikšana var notikt dažādos veidos, gan klātienē - aizpildot pieteikumu, gan attālināti – izmantojot mobilās vai tīmekļa aplikācijas , vai izmantojot apdrošinātāja zvanu centra pakalpojumus.

1.1.2. Zaudējumu pieteikumu reģistrēšana izmantojot saskaņoto paziņojumu

Saskaņotais paziņojums ir standartizēta veidlapa, kurā tiek aprakstīti ceļu satiksmes negadījuma apstākļi **Error! Reference source not found.**. Šīs veidlapas darbojas vairāk kā 20 E iropas valstīs. Saskaņotais paziņojums sastāv no divām veidlapām, kuras aizpilda abi ceļu satiksmes negadījuma dalībnieki. Atbilstoši LTAB datiem **Error! Reference source not found.Error! Reference source not found.**, vairāk kā 64% negadījumu tiek pieteikti, izmantojot saskaņoto paziņojumu. Kopā ir jāaizpilda 15 punkti un jāapliecina ar parakstu. Saskaņotais paziņojums ļauj fiksēt negadījuma apstākļus bez ceļu policijas iesaistes, kas ļauj ekonomēt ceļu satiksmes negadījumu laiku. Šo pieteikuma formu drīkst izmantot, ja CSNg ir iesaistīti tikai 2 transportlīdzekļi, nav cietušas personas, nav cietusi trešo personu manta un negadījumā iesaistītie transportlīdzekļi drīkst un var turpināt ceļu pašu spēkiem.



1.2. att. **Saskaņotā paziņojuma veidlapa**

Tā kā veidlapa ir standartizēta, tad tās digitalizācijai ir iespēja, ar augstu precizitāti, izmantot attēlu atpazīšanas mašīnmācīšanās algoritmus. Ceļu satiksmes negadījumu apstākļi paziņojumā tiek fiksēti grafiska zīmējuma veidā ar aprakstu teksta formā, līdz ar to ir nepieciešams izmantot arī tekstu atpazīšanas algoritmus. Ja ceļu satiksmes negadījums noticis ārzemēs, tad viena no veidlapām var būt aizpildīta kādā no Eiropas valstu valodām. Šajā gadījumā ir jāizmanto gan attēla un teksta atpazīšana, un tulkošana. Visas šīs problēmas ir iespēja risināt ar dziļās mašīnmācīšanās algoritmiem, tādā veidā nodrošinot datu apstrādes automatizāciju.

Sākot ar 2018. gada novembri ir pieejama mobilā aplikācija, lai aizpildītu saskaņotā paziņojuma datus. Tādā veidā jau lielākā daļa datu tiek iegūta strukturētā veidā, bet faktiskie transportlīdzekļu bojājumi tiek iesniegti attēlu veidā. Šobrīd lielākoties šo attēlu informācija tiek apstrādāta manuāli. Automatizēt šo procesu ir liels izaicinājums, bet tas būtiski paātrinātu zaudējuma regulēšanas procesu, taupītu cilvēk resursus un paaugstinātu klientu apmierinātības līmeni.

1.1.3. Zaudējumu pieteikumu reģistrēšana zvanu centrā

Vēl viena iespēja pieteikt zaudējumu, ir uzņēmuma zvanu centrā. Sarunas laikā zvanu centra speciālistam, atbilstoši izstrādātajiem vedņiem, ir nepieciešams iegūt sekojošu informāciju:

- Identificēt klientu
- Personas vārds, uzvārds un personas kods
- dzimšanas datums nerezidentām personām
- Ceļu satiksmes negadījuma datums, laiks un vieta
- Iesaistītie transportlīdzekļi
    - Reģistrācijas numurs
    - Apliecības numurs
    - Marka, modelis, šasijas numurs ārvalstīs reģistrētiem transportlīdzekļiem
- Notikuma apstākļi
- Apdrošināšanas polises numurs (nav obligāti)

Pilna pieteikuma informācija ir pieejama audio informācijas veidā, līdz ar to datu apstrādē ir iespējams izmantot virtuālos asistentus un audio datu analīzi un atpazīšanu. Tādā veidā ir iespējams gan automatizēt zaudējumu pieteikšanas procesu, gan uzkrāt un analizēt klientu apkalpošanas kvalitātes datus. Šāda prakse jau tiek lietota vienā no lielākajām apdrošināšanas kompānijām pasaulē Munich Re grupā, dažos pilotprojektos **Error! Reference source not found.Error! Reference source not found.Error! Reference source not found.**

1.1.4. Zaudējumu pieteikumu reģistrēšanas citas iespējas

Praksē, apdrošināšanas uzņēmumi izmanto kompleksas metodes kā savākt datus, gan klātienē, gan attālināti, gan verbāli, gan izmantojot tehnoloģiskus rīkus. Turpmāk šajā darbā tiks izmantoti jau apkopoti un daļēji strukturēti dati par ceļu satiksmes negadījumiem Latvijā vairāk kā 10 gadu periodā.

1.2. Krāpniecības gadījumu noteikšana

Apdrošināšanas nozarē, viena no jomām, kurai tiek pievērsta liela uzmanība, ir krāpniecības novēršana. Krāpniecība ietekmē ne tikai apdrošināšanas uzņēmumus, bet arī apdrošinātāju klientus, jo krāpniecības gadījumu līmenis ietekmē arī apdrošināšanas prēmiju pieaugumu. Apdrošināšanā, krāpniecība var notikt ļoti daudzos apdrošināšanas procesa posmos – gan risku izvērtēšanas laikā, gan līguma izdošanas procesā, gan veicot maksājumus, gan piesakot un regulējot zaudējumus. To var veikt gan klienti, gan apdrošināšanas starpnieki, brokeri, darbinieki kā arī trešo pušu pakalpojumu sniedzēji – tādi kā transportlīdzekļu remont servisi, vērtētāji, u.c. Apdrošināšanā, draudus var iedalīt 2 grupās, kuriem pēc savas būtības ir pilnīgi dažāda daba. Oportūnistiskā krāpniecība – tos parasti rada privātas personas, kurai ir radusies iespēja iegūt augstāku zaudējuma novērtējumu, piemēram iegūstot augstāku remontdarbu tāmi. Šie krāpšanas gadījumi parasti ir vienkārši un radītie zaudējumi salīdzinoši zemi. Otrā grupa ir profesionālā krāpniecība, kur darbība tiek veikta organizētās grupās un vērsta uzreiz pret vairākiem uzņēmumiem. Lai gan krāpšanas gadījumu skaits šajā grupā ir zemāks, toties izkrāptā summa vienā incidentā ir krietni lielāka.

Salīdzinot ar citiem apdrošināšanas veidiem, transportlīdzekļu apdrošināšanā, krāpniecībai tiek pievērsta īpaši liela nozīme. Aptuveni viena trešdaļa no datizraces un prognozējošās modelēšanas pielietojumiem finanšu draudu monitorēšanas jomā, tiek izmantota transportlīdzekļu apdrošināšanā [6] A. Abdallah,M.A.Maarof, A. Zainal, Fraud detection system: a survey, Journal of Network and Computer Applications 68 (2016) 90–113. https://datubazes.lanet.lv:2076/science/article/pii/S1084804516300571

[7] E.W.T. Ngai, Y. Hu, Y.H.Wong, Y. Chen, X. Sun, The application of data mining techniques in financial fraud detection: a classification framework and an academic review of literature, Decision Support Systems 50 (3) (2011) 559–569. https://datubazes.lanet.lv:2076/science/article/pii/S0167923610001302

[8] Vipula Rawte, G Anuradha 2015 International Conference on Communication, Information & Computing Technology (ICCICT), Jan. 16-17

https://www.researchgate.net/publication/282538462_Fraud_detection_in_health_insurance_using_data_mining_techniques
[9] FRISS [Tiešsaite] https://knowledge.friss.com/survey-insurance-fraud-2019 https://www.friss.com/resources/
[10] Y. Wang, W. Xu, Leveraging deep learning with LDA-based text analytics to detect automobile insurance fraud, Decision Support Systems 105 (2018) 87–95 https://datubazes.lanet.lv:2076/science/article/pii/S0167923617302130
[11] Ibiwoye, A., Ajibola, O. O. E. and Sogunro, A. B. 2012. Artificial Neural Network Model for Predicting Insurance Insolvency, International Journal of Management and Business Research, 2(1) 59- 68.

. Biežāk sastopamie krāpniecības veidi ir:

- Mākslīgi veidotas transportlīdzekļu zādzības
- Negadījuma apstākļu viltošana, sūdzību iesniegšana, nepatiesu zaudējuma apmēra norādīšana pieteikumā, lai no apdrošināšanas gadījuma gūtu lielāku labumu ne kā ir patiesie zaudējumi
- Pieprasīt atlīdzību par trešo pušu atbildību
- Sadarbība ar apdrošināšanas starpniekiem un brokeriem, lai veidotu viltotus zaudējuma pieteikumus

Apdrošināšanas industrijā tiek pielietotas vairākas pamat metodes, lai atklātu krāpniecības gadījumus kā arī, lai tos novērstu nākotnē. [8] Vipula Rawte, G Anuradha 2015 International Conference on Communication, Information & Computing Technology (ICCICT), Jan. 16-17

https://www.researchgate.net/publication/282538462_Fraud_detection_in_health_insurance_using_data_mining_techniques
[9] FRISS [Tiešsaite] https://knowledge.friss.com/survey-insurance-fraud-2019 https://www.friss.com/resources/
[10] Y. Wang, W. Xu, Leveraging deep learning with LDA-based text analytics to detect automobile insurance fraud, Decision Support Systems 105 (2018) 87–95 https://datubazes.lanet.lv:2076/science/article/pii/S0167923617302130
[11] Ibiwoye, A., Ajibola, O. O. E. and Sogunro, A. B. 2012. Artificial Neural Network Model for Predicting Insurance Insolvency, International Journal of Management and Business Research, 2(1) 59- 68.

. Tā kā neeksistē viena labākā metode, praksē tiek izmantotas vairākas metodes kopā kā komplekss risinājums.

1.2.1. Biznesa nosacījumu pārbaudes

Uz nosacījumu pārbaudēm balstītas sistēmas pārbauda transakciju atbilstību iepriekš definētiem likumiem vai algoritmiem. Ja kāds no likumiem nostrādā, tad atrastā transakcija tiek atzīmēta kā aizdomīga. Piemēram, ja kādā noteiktā laika vienībā, atlīdzību skaits uz vienu klientu vai apdrošināto objektu pārsniedz noteiktu slieksni, tad objekts, klients vai transakcija var tik marķēta kā aizdomīga un tālāk var tikt veiktas vai nu automātiskas vai manuālas, risku mazināšanas procesā paredzētās darbības. Šīs metodes priekšrocība ir tās vienkāršība, jo nosacījumi ir stingri definēti, tos ir salīdzinoši viegli saprast. Marķētās transakcijas ir viegli atrast izmantojot datu bāzu meklēšanas iespējas. Šis metodes trūkums ir lielais, nepareizi detektēto krāpniecības gadījumu skaits, kā arī tas, ka pārbaužu nosacījumi ir balstīti uz pagātnes datiem un tādā veidā ir grūti atklāt jaunus krāpniecības veidus. Neskatoties uz šīs metodes trūkumiem, biznesa nosacījumu pārbaudes ir pietiekami labas, pamat aizsardzības nodrošināšanai.

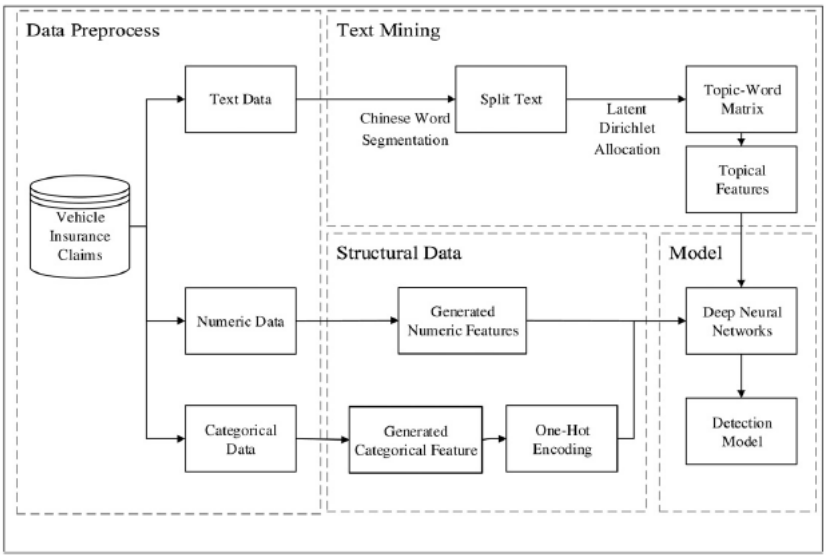
1.2.2. Anomāliju atklāšana

Lai atklātu anomālijas, kas liecina par krāpšanās gadījumiem, ir jānosaka uzdevumu, procesu, notikumu izpildes pamat rādītāji (KPI). Jānodefinē rādītāju sliekšņi normālas uzvedības situācijās. Ja kāda rādītāja slieksnis tiek pārkāpts, tas tiek uzskatīts par anomāliju un attiecīgi tiek marķēts sistēmā vai nosūtīts manuālām pārbaudēm. Šādas pieejas priekšrocība ir tas, ka to ir salīdzinoši viegli ieviest. Ja vajadzīgie sliekšņi un pamat rādītāji ir noteikti, sistēma var darboties automātiskā režīma un zaudējumu speciālistu uzdevums ir sekot līdzi atklātajām anomālijām un rīkoties atbilstoši noteiktajām procedūrām. Sarežģītākā šīs pieejas daļa ir noteikt indikatorus, ko mērīt, noteikt pareizus sliekšņus, lai noteiktu anomālijās, bet tajā pašā laikā, lai minimizētu nepareizi noteikto incidentu skaitu. Nosakot sliekšņus par augstu, netiks noteikti potenciālie krāpniecības gadījumi, savukārt nosakot tos par zemu, tiks tērēti resursi, lai izskatītu nepareizi noteiktos krāpniecības gadījumus. Anomāliju atklāšanā bieži tiek lietotas divi analīzes veidi: profilēšanas modeļi, kur tiek izveidoti personu vai grupu uzvedības modeļi balstoties uz vēsturiskajiem datiem un pēc tam tiek pielietoti uz aktuālajiem datiem. Otra pieeja ir klasteru noteikšana.

1.2.3. Prognozējošā modelēšana

Pēdējos gados apdrošināšanas uzņēmu arvien vairāk sāk veidot datu prognozēšanas modeļus, lai risinātu dažādus, ar apdrošināšanas problēmām saistītus, jautājumus. Viens no veidiem kā veidot šādus modeļus ir izmantojot mašīnmācīšanās metodes. Mašīnmācīšanās metodēm ir vairākas priekšrocības salīdzinot ar citām pieejām. Viena no tām ir ātrums. Tā kā

liela daļa no operācijām notiek tiešsaistē, ir svarīgi pieņemt lēmumus ātrāk kā sekundes laikā un katras transakcijas vai operācijas laikā ir svarīgi pieņemt lēmumu par krāpniecības gadījuma iespējamību. Apmācīts mašīnmācīšanās modelis spēj to paveikt bez manuālas iesaistes. Ir jāņem vērā arī tas, ka ir jāapstrādā ne tikai strukturēti dati, bet arī tekstuāla informācija, audio dati, kā arī vizuāla informācija, piemēram transportlīdzekļu bojājumi dažādos laika brīžos. Zemāk redzamajā attēlā ir redzams vispārējs krāpniecības noteikšanas ietvars.



1.3. att. **DNN ietvars apdrošināšanas krāpniecības noteikšanai [10]**

Otra priekšrocība ir mērogojamība. Tā kā datu apjoms nemitīgi aug, modeļa veiktspēju ir iespējams nepārtraukti uzlabot. Atšķirībā no uz likumiem un nosacījumiem balstītām sistēmām, ar mašīnmācīšanās modeļiem ir iespēja vispārināt šos noteikumus uz visu datu kopu kā arī piemērot to jaunām transakcijām. Trešā joma ir efektivitātes palielināšana, jo mašīnmācīšanās modelis veikt datu analīzi tiešā laika režīmā bez cilvēk resursu iesaistes. Nepārraudzītie mašīnmācīšanās modeļi nodrošina nepārtrauktu modeļa analīzi un uzlabošanu, tādējādi nevajag manuāli veidot jaunus nosacījumus un incidentu detektēšanas likumus.

Modeļa izveides process sastāv no vairākām daļām [11] . Galvenās no tām ir: Datu iegūšana un apstrāde, treniņa, testa un validācijas datu kopu izveide un paša modeļa izstrāde. Būtiski i pievērst uzmanību datu nepilnībām un kļūdām. Ja tās netiek izslēgtas no modeļa, tas ietekmē izstrādātā modeļa veiktspēju. Veidojot modeli ir jārisina divu veidu uzdevumi: regresijas un klasifikācija. Parasti regresijas uzdevumus var risināt ar klasifikācijas uzdevumu risināšanas metodēm, ieviešot ciparisku vērtību intervālus un izveidojot klases. Piemēram, ja skatāmies apdrošināšanas atlīdzību summas un gribam prognozēt nākotnes vērtības, tad varam tās sadalīt intervālos:

- Līdz 300 Eur
- 300-1000 Eur

- 1000-10000 Eur
- Vairāk par 10000 Eur

Rezultātā tiek iegūtas 4 klases:

- I Mazas atlīdzības
- II Vidējas atlīdzības
- III Lielas atlīdzības
- IV Ļoti lielas atlīdzības

Liela nozīmē tam cik labi darbosies modelis ir modeļa pazīmju (features) izveidei un izvēlei. Ir būtiski izvēlēties tās pazīmes, kas pēc iespējas labāk raksturo apskatāmo problēmu. Pazīmju izveide ir iteratīvs process. Kas sastāv no datu novērtēšanas, pazīmju izveides un izvēles, modeļa rezultātu analīzes. Ja iegūtais rezultāts nav apmierinošs, tad iepriekš minētais process tiek atkārtots. Pazīmju izvēlē galvenokārt darbojas kā filtrēšana, kuras rezultātā tiek atmestie datu atribūti, kas mazina modeļa veiktspēju [11] . Izšķir vairākus modeļa pazīmju izvēles veidus.

Pazīmju skaita palielināšana - tas nozīmē, ka modelis tiek darbināts ar vienu izvēlētu pazīmi, tālāk tiek izvērtēti modeļa rezultāti un pazīmju skaits palielināts. To dara tikmēr kamēr ir sasniegts vēlamais modeļa veiktspējas līmenis vai arī turpmākus uzlabojumus vairs neizdodas iegūt.

Otrs veids ir pazīmju izslēgšana. Tas darbojas tādā veidā, ka modelis sākotnēji darbojas ar visām izvēlētajām pazīmēm, tālāk tiek novērtēti modeļa darbības rezultāti un pazīmju skaits tiek samazināts, kamēr modeļa darbības rezultāti ir vislabākie.

Lai izvēlētos labākās modeļa pazīmes var izmantota ar klasiskas mašīnmācīšanās metodes, piemēram galveno komponentu analīzi (PCA). Ar šis metodes palīdzību, sākotnējie daudzdimensiju dati tiek pārvērsti uz mazāku dimensiju skaitu un tiek izvēlētas tās dimensijas, kuras vislabāk apraksta datu dispersiju.

Datu sagatavošana un analīze, datu kopu izveide, modeļa izstrāde, pazīmju izvēle un veiktspējas uzlabošana tiks apskatīta tālākā darba gaitā.

1.3. Apdrošināšanas risku novērtējums

Risku izvērtēšana ir visu apdrošināšanas uzņēmumu pamat nodarbošanās. No tā cik kvalitatīvi tiek veikts šis uzdevums tiešā veidā ir atkarīgs cik lielu peļņu gūs apdrošinātājs. Ir pašsaprotami, ka ir nepieciešams veikt gan apdrošināmo objektu selekciju, gan apdrošināmo risku izvēli, kā arī jānosaka klienta/objekta riska līmenis. Pretējā gadījumā tie klienti, kas neizraisa zaudējumus būs spiesti maksāt lielāku apdrošināšanas prēmiju nekā vajadzētu un visdrīzāk pāries pie citiem apdrošināšanas uzņēmumiem, kuriem prēmijas aprēķins ir izsmalcinātāks.

Apdrošināšanas industrijā pastā divi galvenie parametri, kuri nosaka klienta līmeni. Tie ir atlīdzību vai apdrošināšanas gadījumu biežums un apdrošināšanas atlīdzību lielums. Ja, zinot apdrošināmā objekta, iesaistīto personu, ārējo apstākļu faktorus, izdotos prognozēt ar pietiekami augstu precizitāti atlīdzību biežumu un lielumu, tas ļautu klientiem piedāvāt pašus optimālākos nosacījumus, kā arī izvairīties no pārāk augsta riska objektiem un personām vai arī iniciēt pārapdrošināšanas procesu.

Ar risku parametru novērtēšanu un aprēķiniem apdrošināšanas uzņēmumos nodarbojas aktuāri, kuri izmanto dažādas metodes un rīkus. Lielākajā daļā gadījumu, aktuāru risinātās problēmas var reducēt uz regresiju problēmu risināšanu, kuras savukārt var veikt ar mašīnmācīšanās palīdzību.

1.3.1. Īstermiņa apdrošināšanas līgumu prēmiju aprēķināšana

Nedzīvības apdrošināšanā parasti līguma ilgums ir 12 mēneši. Protams ir apdrošināšanas veidi, kuros tiek izmantoti daudzgadīgie līgumi, bet raugoties no apdrošināšanas tehnikas viedokļa, bieži vien katrs nākamais gads ir iepriekšējā līguma atjaunojums. Īstermiņa apdrošināšanas līgumi ir tādi līgumi, kuru ilgums ir mazāks nekā 1 gads – sākot no vienas dienas līdz pat vairākiem mēnešiem. Šādu līgumu prēmiju aprēķināšanā tiek izmantots vispārinātais lineārais modelis (GLM) [12] , kur atlīdzību biežuma noteikšanai tiek izmantota Puasona regresija [12] , bet atlīdzību lieluma noteikšanai Gamma regresija. Ieejas pazīmju vektors šiem modeļiem bieži vien satur apdrošinājuma ņēmēja dzimšanas datus, dzimumu, dzīvesvietas adreses informāciju, transportlīdzekļa parametrus (marka, modelis, dzinēja tilpums, jauda, u.c.). Kā izejas dati šiem modeļiem ir vai nu aprēķinātā apdrošināšanas prēmija vai arī prognozētais atlīdzību lielums un/vai biežums

1.3.2. IBNR rezervju noteikšana

Apdrošināšanas uzņēmumos aktuāri uzdevums ir rēķināt pieteikto, bet vēl neizmaksāto atlīdzību rezerves (IBNR). IBNR tiek rēķināts prognozējot potenciālos apdrošināšanas zaudējumus un tas ļauj apdrošinātājam rezervēt finanšu līdzekļus, lai nodrošinātu apdrošināšanas atlīdzību izmaksas, tādejādi rūpējoties, lai uzņēmums būtu maksātspējīgs. IBNR aprēķinos tiek izmantots Baiesa hierarhiskais modelis un vispārējie jaukta tipa lineārie modeļi [13] , Chain-ladder metode [14] Galvenie ieejas parametri šiem modeļiem ir apdrošināšanas negadījuma datums, zaudējuma pieteikuma datums, apdrošināšanas līguma veids, radītais zaudējuma apmērs iepriekšējos laika posmos. Modeļa rezultāts ir prognozētais atlīdzību lielums tālākajos laika posmos. IBNR rezervju aprēķināšana ir regresijas uzdevums un skatoties no datu viedokļa, tie ir izkārtoti trijstūra veidā (sk. 1.1. tabula). Augšējais trijstūris ir zināmie dati, bet apakšējais – tas kas jāprognozē

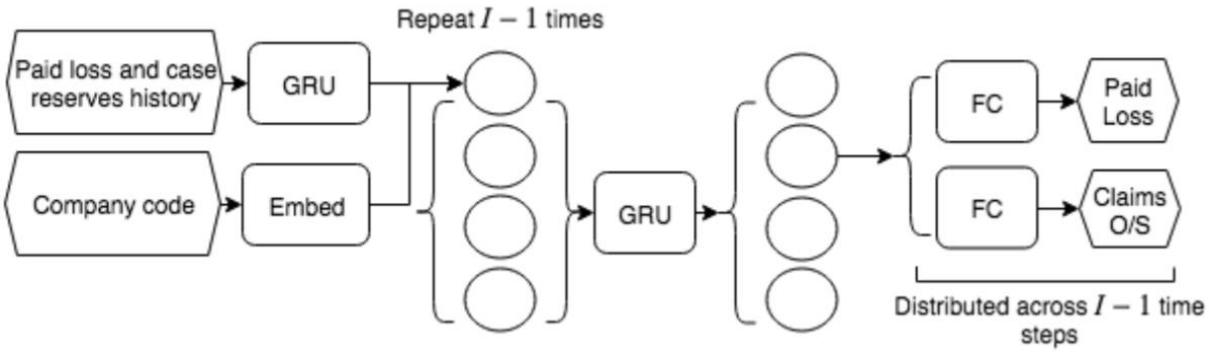
1.1. tabula

Atlīdzību datu paraugs

Atlīdzību gads	Atlīdzību izmaksas (pa gadiem)					
	1	2	3	4	5	6
2010	5000	7000	11000	12000	13500	15000
2011	4500	6900	9000	11000	13600	
2012	5100	8000	9500	12100		
2013	5200	7900	12000			
2014	5100	8100				
2015	6000					

Ja klasiskās IBNR aprēķināšanas metodes izmanto apdrošināšanas atlīdzību kumulatīvos lielumus, tad dziļie neironu tīkli ļauj risināt IBNR aprēķinu problēmas, [15] ņemot vērā katru individuālu atlīdzību gadījumu. Viena no pieejām ir DeepTriangle [15] . Šis neironu tīkla modelis tika veidots par pamatu ņemot apdrošināšanas atlīdzību datus no 50 ASV apdrošināšanas kompānijām. Atlīdzību pieteikumu periods bija no 1988 - 1997 gadam, atlīdzību izmaksas 1998-2006 gadam. Dati satur informāciju par 4 apdrošināšanas biznesa līnijām

Šī tīkla arhitektūra satur tādus komponentus, kā "embedding" slāni, kurā tiek iekodēti apdrošināšanas kompāniju kodi, GRU (Gated recurrent units) komponente, kas ir rekurento neironu tīklu sastāvdaļa, FC – pilnībā saistīto neironu slānis.



1.4.att. **DeepTriangle arhitektūra [15]**

Kā ieejas dati, papildus apdrošināšanas uzņēmuma kodam, ir izmaksātās apdrošināšanas atlīdzības un apdrošināšanas rezervju dati, bet kā izejas dati ir plānotās izmaksātās un aprēķinātās, bet vēl neizmaksātās atlīdzības. Šim tīklam ir daudz uzdevumu arhitektūra, kura galvenais uzdevums ir prognozēt izmaksājamās atlīdzības un papildus noteikt vajadzīgās rezerves katrā laika posmā.

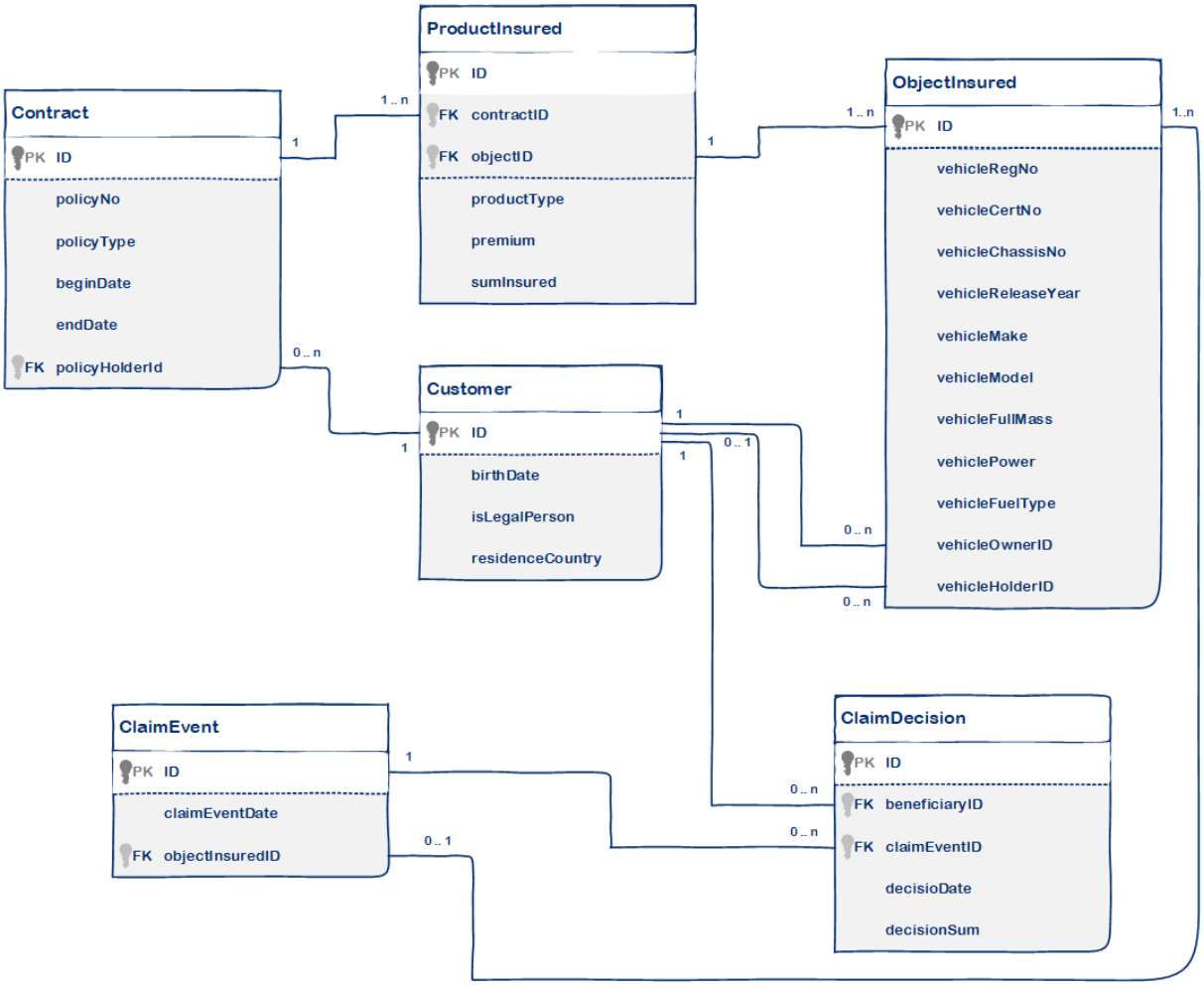
2. APDROŠINĀŠANAS ATLĪDZĪBU DATU APSRĀDE UN ANALĪZE

2.1. Datu avota apraksts

Darbā izmantotie dati tika iegūti no Latvijas transportlīdzekļu apdrošināšanas biroja (LTAB) datu bāzes. LTAB ir apvienojis visas apdrošināšanas sabiedrības, kurām ir tiesības veikt sauszemes transportlīdzekļu īpašnieku civiltiesiskās atbildības obligāto apdrošināšanu Latvijā. Datus šajā datubāzē ieziņo Latvijas apdrošināšanās kompānijas un dati satur informāciju par noslēgtajiem un pārtrauktajiem apdrošināšanas līgumiem, par ceļu satiksmes negadījumiem, par tajos iesaistītajiem transportlīdzekļiem, par ceļu policijas protokoliem, saskaņotajiem paziņojumiem, zaudējumu pieteikumiem, par lēmumu pieņemšanas procesu, u.c. dati. Darbā tiek izmatoti dati, kuri iegūti par periodu no 2008. gada 1. janvāra līdz 2019. gada 1. janvārim

2.1.1. Datu bāzes ER diagramma

Diagrammā ir aprakstītas tās entītijas un atribūti, kuri tiek izmantoti šajā darbā



2.1 att. **ER diagramma**

2.1.2. Datu dimensiju un faktu tabulas

Šajā darbā izmantotie dati atrodas vairākās dimensiju un faktu tabulās. Kopumā tika izmantots viens datu filtrācijas nosacījums, 8 datu dimensiju tabulas un 4 datu faktu tabulas. Galvenās izmantotās datu noliktavas dimensijas ir Apdrošināšanas līguma datu dimensija, kas satur pamatinformāciju par noslēgtajiem apdrošināšanas līgumiem, piemēram, līguma izdošanas datums, sākuma datums, apdrošināšanas līguma ilgums dienās, u.c. Apdrošinātāja datu dimensija satur datus par apdrošināšanas uzņēmumu, tādus kā kods, nosaukums, reģistrācijas nr. u.c. Personu datu dimensija ir sagrupēta 4 daļās un satur datus gan par transportlīdzekļa īpašnieku, gan turētāju, gan BM klases aprēķina subjektu kā arī par transportlīdzekļa vadītāju. Transportlīdzekļa vadātāja dimensiju var izmantot tikai kopā apdrošināšanas zaudējuma pieteikuma un cietušo vainīgo objektu datiem, jo citas entītijas tos nesatur. Personu datu dimensijas satur tādus datu atribūtus, kā personas vārds, uzvārds, identifikators, dzimums, adreses dati, u.c. Transportlīdzekļu datu dimensija satur apdrošināšanas līguma objekta datus, kā arī ceļu satiksmes negadījumā iesaistīto transportlīdzekļu datus. Šie dati satur tādus atribūtus kā transportlīdzekļa marka, modelis, izlaiduma gads, identifikators, pilna masa u.c. Ceļu satiksmes negadījumu dati satur informāciju par negadījuma datumu, laiku, vietu kā arī ir ietverta negadījuma apraksts brīva teksta veidā. Zaudējumu pieteikumu un lēmumu dimensijas nodrošina datu atlasīšanu par zaudējuma pieteikuma un lēmuma datumu un laiku kā arī klasifikatoriem. Maksājuma dimensijā ir iekļauti tādi dati kā maksājuma veids, datums, numurs, u.c.

Šajā darbā tiek izmantotas 4 faktu tabulas: Ceļu satiksmes negadījumi, atlīdzību rezerves, Lēmumi un Maksājumi. Visas šīs faktu tabulas satur datus par skaitu, valūtu un attiecīgas summas norādītajā valūtā

Dati no dimensiju un faktu tabulām tiek filtrēti balstoties uz ceļu satiksmes negadījuma notikuma datumu un laiku.

2.1.3. Datu lauki

Darbā izmantotie datu lauki iedalīti trīs grupās. Ar apdrošināšanas līgumu saistītie datu lauki:

- AAS nosaukums
- BM subjekta adreses ATVK kods
- BM subjekta dzimšanas datums
- BM subjekta ISN
- BM subjekta personas veids
- Līguma noslēgšanas kanāls
- Polises ilgums (dienās)
- Polises sākuma datums

Otra grupa ir transportlīdzekļa un tā vadītāja dati:

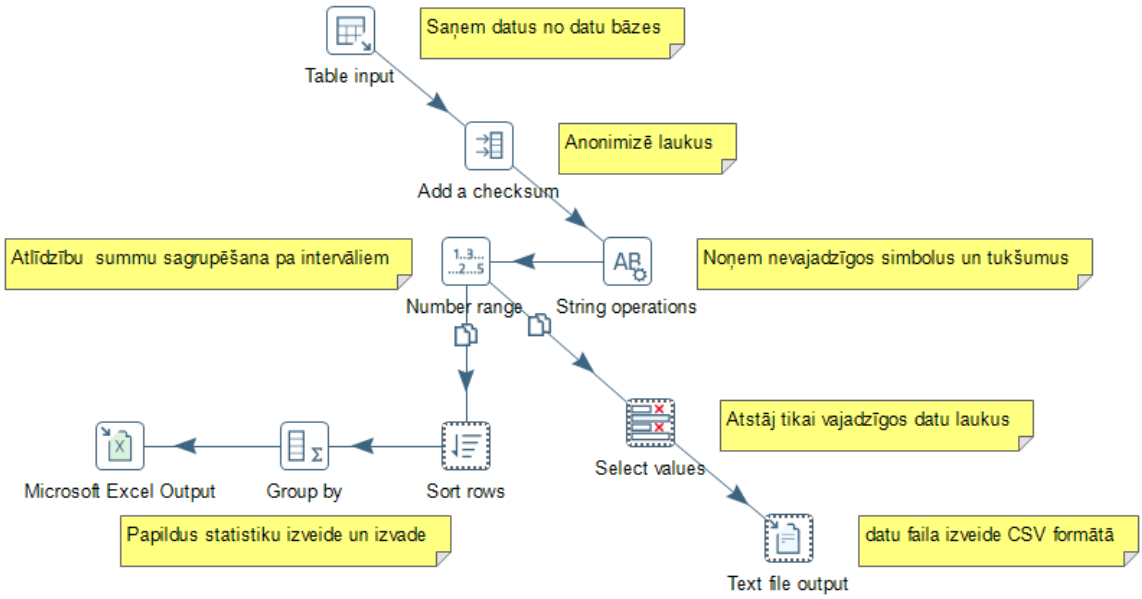
- ISN
- Izlaides gads
- Dzinēja tilpums cm2
- Degvielas tips
- Dzinēja jauda
- Pilna masa
- Krāsa
- Marka
- Modelis
- TRL kods

Trešajā grupā ir ietverti ar ceļu satiksmes negadījumu un atlīdzību izmaksām saistītie dati:

- CSNg apraksts
- CSNg datums
- CSNg iemesls
- Lēmuma pieņemšanas datums
- Ir ilgtermiņa lieta
- Pieteikuma datums
- Pieteikuma ID
- Lēmuma veids
- CSNg novada ATVK kods
- Rezerves summa (EUR)
- Lēmuma summa (EUR)

2.2. Datu analīzē izmantotie rīki

Dati tika iegūti, izmantojot SAP BusinessObjects datu noliktavas risinājumu. Tika izmantota sistēmas versija 4.2. Datu pareizība un atbilstība tika pārbaudīta veicot SQL vaicājumus no Oracle 12c datubāzes. Dati tika iegūti izmantojot Oracle veidoto rīku SQL developer versija 18.3. Datu transformācijām tika izmantots Pentaho Data integration rīks versija 8.0. Pentaho risinājums tika izmantots, lai veiktu datu priekšapstrādi un transformācijas. Izmantoto ETL skriptu shēmu var aplūkot attēlā.



2.2.att. **Datu transformāciju shēma**

Tālāk datu apstrāde un mašīnmācīšanās modeļa izveide tika veikta izmantojot Python programmēšanas valodu Python. Python valoda tika izvēlēta, jo tā satur lielu skaitu dažādu datu analīzē un apstrādē izmantojamo bibliotēku kā arī atbalsta Tensorflow ietvara izmantošanu.

2.3. Datu aizsardzība un maskēšana

Lai apdrošināšanas uzņēmumi varētu klientiem sniegt apdrošināšanas pakalpojumus, tiem ir nepieciešami klientu personas dati. Klientu personas dati tiek apstrādāti, lai varētu piedāvāt labākos apdrošināšanas risinājumus, izpildīt apdrošināšanas līgumos un izpildīt juridiskās prasības. Personas datu aizsardzība ir viena no apdrošinātāju prioritātēm un tāpēc tie nodrošina klientu personas datu konfidencialitāti saskaņā ar likuma prasībām un, apstrādājot klientu personas datus, tiek pielietoti dažādi organizatoriskie un tehniskie pasākumi.

2.3.1. Likumi un regulas

Personu datu aizsardzības principus Latvijā nosaka Fizisko personu datu apstrādes likums, kas nosaka Vispārīgās datu aizsardzības regulas (GDPR) piemērošanu Latvijā. Regulas piemērošana tika uzsākta 2018. gada 25. maijā un tā definē vienotus personu datu aizsardzības principus Eiropas savienībā. Regula nosaka vienotus nosacījumus personas datu aizsardzībai, kas attiecināmi uz apstrādi, uzturēšanu, nodošanu citiem uzņēmumiem un arhivēšanu, un vienotus noteikumus visām kompānijām, neskatoties uz to reģistrācijas valsti. Uzņēmumiem, kuri neievēro šīs likuma normas var tikt piemērotas ievērojamas soda sankcijas, līdz ar to apdrošinātājiem veicot automatizāciju rūpīgi jāizvērtē riski, kas saistīti ar datu apstrādi. Labā ziņa, ka lai apmācītu neironu tīklus vai arī izmantot mašīnmācīšanās algoritmus ir pietiekami izmantot kodētu datus, kuros ir saglabātas datu īpašības, piemēram unikalitāte.

2.3.2. Datu maskēšana

Datu maskēšanas tehnoloģijas nodrošina to aizsardzību pret nesankcionētu lietošanu, tādā veidā, ka datu sensitīvās pazīmes tiek noņemtas, bet dati tomēr saglabā savas strukturālās īpašības. Praksē tiek lietotas dažādas datu maskēšanas metodes [16] .

- Kriptēšana - dati tiek kriptēti ar kādu no kriptēšanas algoritmiem
- Simbolu mainīšana - datu laukā teksta simboli tiek samainīti vietām
- Aizvietošana - datu lauku vērtības aizvietota ar citām vērtībām

Pieejamajā datu bāzē atrodas aktuāli dati ne vecāki par 2008. gada 01. janvāri. Tiek izmantoti tikai ar transportlīdzekļu apdrošināšanas veidiem saistītie apdrošināšanas līgumu, zaudējumu regulēšanas un personu dati Personas dati, pēc kuriem var identificēt personu, kā arī apdrošināšanas līguma dati, pēc kuriem var identificēt līgumu tiek maskēti. Lauku vērtību maskēšanai tiek izmantota kriptēšanas un aizvietošanas metode. Dati tiek kriptēti ar SHA-256 algoritms - lauku vērtības tiek zaudētas, bet vērtības unikalitāte tiek saglabāta.

2.1. tabula

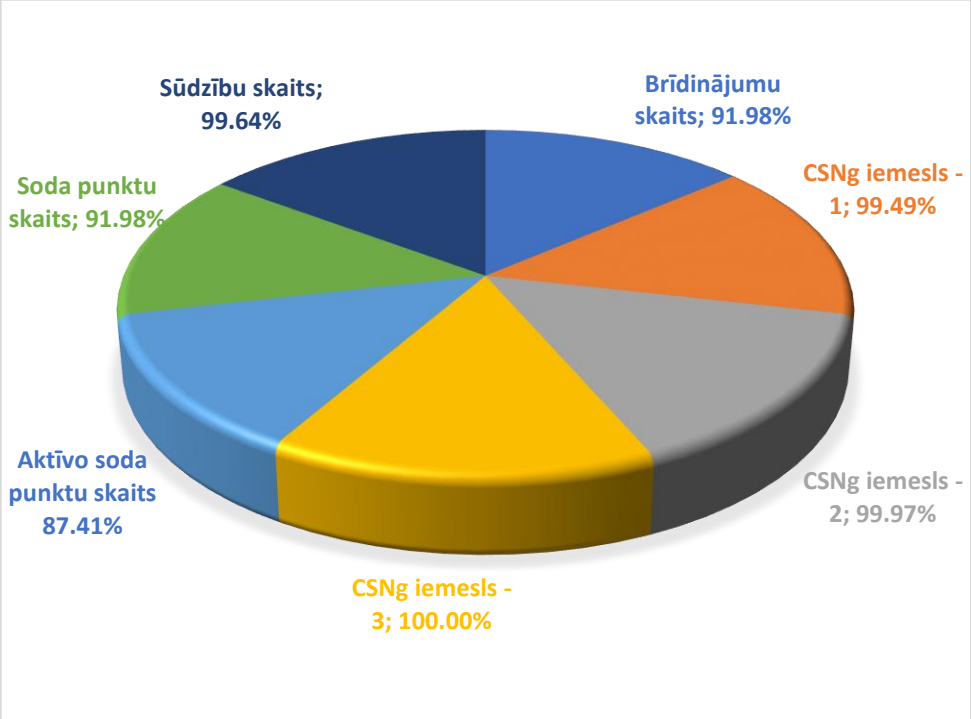
Maskētie datu lauki

Lauks	Darbība
Juridiskas personas nosaukums	Anonimizēts
Vārds	Anonimizēts
Uzvārds	Anonimizēts
E-pasts	Anonimizēts
Dzimšanas datums	Anonimizēts - tiek atstāta tikai mēneša un gada informācija, diena tiek aizvietota ar 01 (piemēram 1993.08.23 aizvietots ar 1993.08.01)
Fiziskas personas kods / Juridiskas personas identifikators	Anonimizēts
Transportlīdzekļa Identifikators	Nemainīts
Transportlīdzekļa reģistrācijas Nr.	Aizvietots ar identifikatoru
Transportlīdzekļa reģistrācijas apliecības Nr.	Aizvietots ar identifikatoru
Transportlīdzekļa šasijas Nr.	Aizvietots ar identifikatoru

2.4. Datu kvalitātes novērtējums un apstrāde

2.4.1. Datu lauku selekcija

Darbā izmantotie dati satur 49 datu kolonas un 618737 datu rindas. Analizējot datus tika secināts, daudzi lauki satur nozīmīgu skaitu nan vērtības. (3. 3. pielikums. Datu kolonu python tipi un netukšās vērtības), tāpēc šos laukus nav nozīmes izmantot tālākā modeļa apmācībā. Jāņem vērā, ka datu kopa satur ne tikai jau izmaksātās apdrošināšanas atlīdzības, bet arī tādus zaudējumu pieteikumus, kas vēl nav izmaksāti vai arī, kuros tika pieņemts lēmums atteikt zaudējumu atlīdzināšanu. Līdz ar to no datu kopas tika izņemti visi tie ieraksti, kas satur nan vērtības. Izmaksāto atlīdzību summas datu failā tika glabātas kā teksts, kas saturēja tukšumus un kā decimālais atdalītājs tika lietots komats, tāpēc šie dati tika modificēti, noņemot tukšumus un tika veikta datu konvertācija uz float64 datu tipu. Orģinālie lauku nosaukumi satur tukšumus un tā kā Tensorflow estimatoru bibliotēkās ir ierobežojumi uz tukšumu lietošanu lauku nosaukumos, tāpēc tie ir aizvietoti ar angliskajiem nosaukumiem.



2.3. att. **Tukšo vērtību īpatsvars datos**

Apskatot atsevišķus laukus secināju, ka lauks "AAS kods" satur tikai vienu vērtību - "Cits apdrošinātājs", tāpēc šis lauks turpmāk netiks ņemts vērā. Laukiem "CSNg iemesls - 1" ,"CSNg iemesls - 2" un "CSNg iemesls - 3" vairāk kā 99% vērtības ir tukšas. Šie lauki satur negadījuma aprakstu brīva teksta formā un tā kā šajā darbā nav ietverta atlīdzību pieteikumā savāktās tekstuālās informācijas analīze, tāpēc šie lauki tika izņemti no apstrādājamās datu kopas.

Ņemot vērā augstāk minēt, turpmāk pētījumā tiks izmantoti sekojoši datu lauki: "Lēmuma summa (EUR)", "Marka", "Modelis", "Izlaides gads", "Pilna masa", "Dzinēja jauda", "Krāsa", "TRL kods", "BM subjekta dzimšanas datums", "BM subjekta dzimums", "BM subjekta personas veids", "BM subjekta adreses ATVK kods", "CSNg datums", "CSNg novada ATVK kods", "ACCIDENT_ATVK", "Pieteikuma datums", "Polises izdošanas datums", "Polises sākuma datums", "Polises ilgums (dienās)", "Lēmuma veids". Tā kā ne visas Python un Tensorflow ietvara bibliotēkas atbalsta nosaukumus unicode kodējumā un tukšuma zīmes, tāpēc izmantotie datu lauki tika pārsaukti izmantojot angliskos nosaukumus un tukšuma zīmes aizvietojot ar pasvītrojuma zīmes simbolu (sk. 2.2. tabula) Datu lauku nozīmi skatīt 0. 2. pielikums. Datu noliktavas datu lauku apkopojums.

2.2. tabula

Datu lauku nosaukumu kartēšana

Oriģinālais nosaukums	Jaunais nosaukums
Lēmuma summa (EUR)	DECISION_SUM_EUR
Marka	VEHICLE_MAKE
Modelis	VEHICLE_MODEL
Izlaides gads	VEHICLE_ISSUE_YEAR
Pilna masa	VEHICLE_FULL_MASS
Dzinēja jauda	VEHICLE_POWER
Krāsa	VEHICLE_COLOR
TRL kods	VEHICLE_CODE
BM subjekta dzimšanas datums	BM_BIRTH_DATE
BM subjekta dzimums	BM_GENDER
BM subjekta personas veids	BM_PERSON_TYPE
BM subjekta adreses ATVK kods	BM_PERSON_ATVK
CSNg datums	ACCIDENT_DATE
CSNg novada ATVK kods	ACCIDENT_ATVK
Pieteikuma datums	ACCIDENT_REPORTING_DATE
Polises izdošanas datums	POLICY_ISSUE_DATE
Polises sākuma datums	POLICY_BEGIN_DATE

Oriģinālais nosaukums	Jaunais nosaukums
Polises ilgums (dienās)	POLICY_LENGTH
Lēmuma veids	DECISION_TYPE

2.4.2. Datu lauku apstrāde

2.4.2.1. Datumu lauki

Apskatot datumu laukus, izrādījās, ka dažādos datumu laukos datumi tiek glabāti dažādā formātā, piemēram "%d.%m.%Y" un "%y.%d.%m", kur %d - mēneša diena attēlota ar divām zīmēm, %m - mēnesis, kas attēlots ar divām zīmēm, %Y - gads, attēlot kā četrciparu skaitlis. %y - gads, attēlots kā divciparu skaitlis. Tā kā datumu lauki no datu faila ir ielādēti kā teksts tie tika konvertēti uz datuma formātu. Lai šos datuma laukus varētu izmantot pazīmju vektoros, lai apmācītu gan regresijas, gan klasifikācijas neironu tīkla modeļos, šie lauki tika sadalīti kā atsevišķi skaitļu tipa lauki, kas satur gada, mēneša un dienas informāciju. Piemēram, lauks "ACCIDENT_DATE", kas satur apdrošināšanas negadījuma datumu, tika sadalīts trīs atsevišķos laukos "ACCIDENT_DATE_YEAR", "ACCIDENT_DATE_MONTH", "ACCIDENT_DATE_DAY".

2.4.2.2. Skaitļu tipa datu lauki

Teksta laukiem, tādiem kā transportlīdzekļa masa, izlaides gads, transportlīdzekļa jauda, atlīdzību summa un ATVK informāciju saturošie lauki, kuri pēc savas nozīmes satur skaitliskas vērtības, tika izdzēstas tukšuma zīmes un aizvietots decimālais atdalītājs ar punktu un vērtības konvertētas uz skaitļu tipu.

2.4.2.3. Kategorizētie datu lauki

Tālāk tika analizēti datu lauki, kas satur kategorizētus datus. Tie ir transportlīdzekļa marka, modelis, krāsa, LTAB kods, BM subjekta dzimums, Lēmuma veids un BM subjekta personas tips. Lai tālāk būtu iespējams veidot pazīmju kolonas ir svarīgi saprast kādas vērtības šie lauki satur un kādās un cik lielās grupās tās dalās. Kā arī ir būtiski zināt null vērtību īpatsvaru, lai varētu izvērtēt vai šie dati ir vispār lietojami tālākā darba izpildē.

2.4.2.4. Transportlīdzekļa marka

Transportlīdzekļu markas lauks VEHICLE_MAKE satur 1519 dažādas vērtības, daļa no tām gan ir saistīta ar kļūdainu datu ievadi vai arī datu apmaiņas un datu konvertācijas problēmām starp apdrošinātāju, LTAB un CSDD, kā piemēram "ŠKDA", "ŠKODA", "Š□KODA". Šo nepareizo vērtību īpatsvars ir ļoti neliels – mazāks kā 0.5% un tas būtiski neietekmē kopējo datu kvalitāti. Null vērtību īpatsvars šajos datos ir mazāks kā 1,4%.

2.4.2.5. Transportlīdzekļa modelis

Transportlīdzekļa modeļa lauks VEHICLE_MODEL satur 7142 dažādas modeļu vērtības. Null vērtību īpatsvars datos ir aptuveni 3%. Lielāko ietekmi uz null vērtībām un nekorektām modeļu vērtībām veido ieraksti, kas ir saņemti par traktortehniku. Tas ir izskaidrojams ar to, ka traktortehnikas un lauksaimniecības tehnikas dati tiek pārraudzīti nevis CSDD, bet Valsts tehniskās uzraudzības aģentūras datu bāzēs, kur tiek izmantoti atšķirīgi tehniskie risinājumi, kas vēsturiski ir ietekmējuši arī datu kvalitāti.

2.4.2.6. Transportlīdzekļa krāsa

Transportlīdzekļa krāsas lauks VEHICLE_COLOR satur 322 dažādus ierakstus, bet datu kvalitāte ir augsta, jo dati ir veidoti atbilstoši CSDD datu bāzē esošajam krāsu klasifikatoram. Null ierakstu skaits sasniedz 8%, bet tas ir saistīts ar traktortehnikas un lauksaimniecības tehnikas reģistra datu kvalitāti.

2.4.2.7. Transportlīdzekļa kods

Transportlīdzekļa koda lauks satur informāciju par to pie kādas transportlīdzekļu grupas transportlīdzeklis pieder - vieglais, kravas, traktortehnika, motocikls, u.t.t. Kā arī tiek ņemta vērā transportlīdzekļa kravnesība un jauda. Šis klasifikators ir izstrādāts LTAB un apstiprināts ar Ministru kabineta noteikumiem Nr.801 "Noteikumi par sauszemes transportlīdzekļu īpašnieku civiltiesiskās atbildības obligātās apdrošināšanas informācijas sistēmas darbībai nepieciešamo datu apjomu un veidiem, datu ievades, apmaiņas un izmantošanas kārtību". Tā kā šis lauks ir obligāts datu apmaiņā ar LTAB tad null vērtību datos nav un visas vērtības ir atbilstoši klasifikatoram. Kopējais grupu skaits ir 59.

2.4.2.8. BM subjekta dzimums

BM subjekta dzimumu lauks satur trīs iespējamās vērtības "S" - sieviete, "V" – vīrietis, "n/z" – vērtība nav zināma. Null vērtību īpatsvars ir aptuveni ~0,8%. Dzimumu sadalījumu

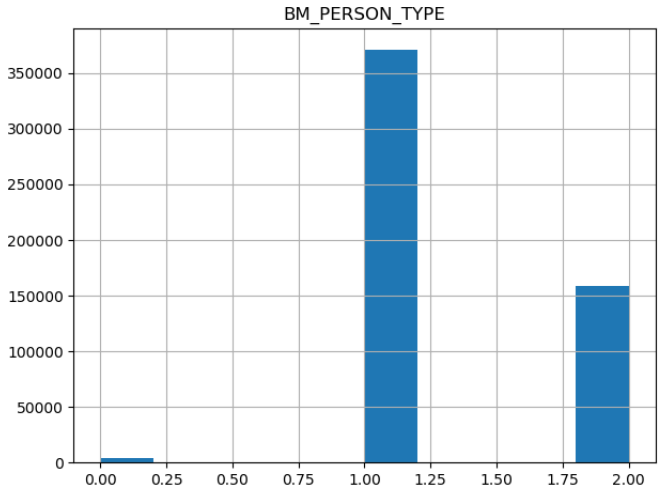
2.3. tabula

BM subjekta dzimumu sadalījums

Dzimums	Skaits
S	143732
V	262282
n/z	206995

2.4.2.9. BM subjekta personas tips

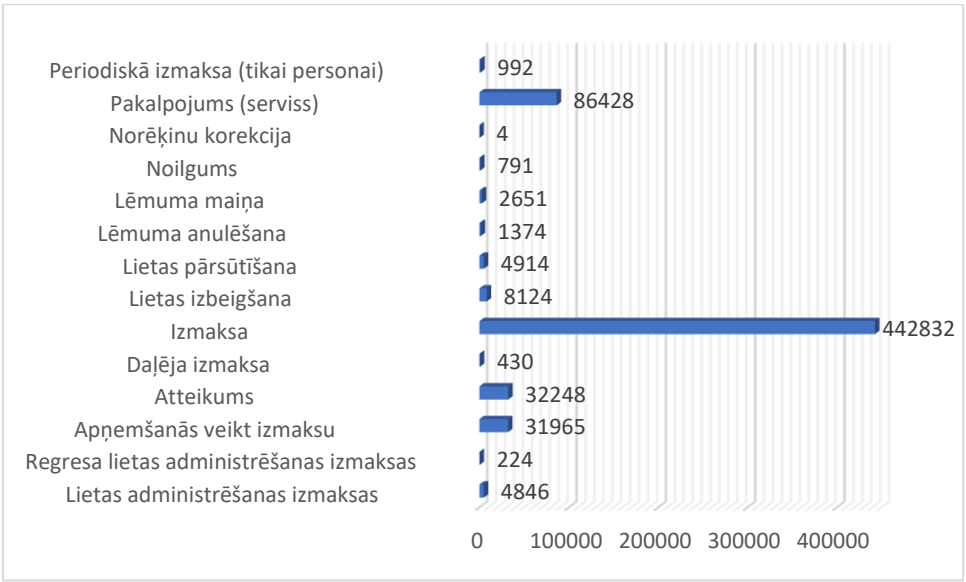
BM subjekta personas tipa lauks BM_PERSON_TYPE satur datus par juridisku un fizisku personu sadalījumu. Lauks satur tekstuālas vērtības: “(nav zināms)“, “Fiziska persona”, “Juridiska persona”. Null vērtību īpatsvars ir ļoti zems - mazāks par 0.8%



2.4. att. **Personas tipu sadalījums**

2.4.2.10. Lēmuma veids

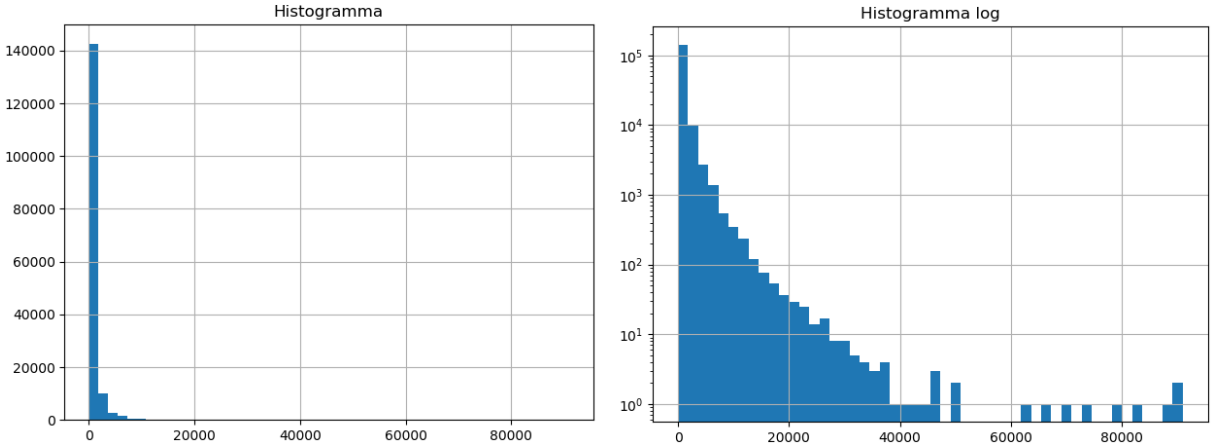
Atlīdzību lēmuma veida lauks DECISION_TYPE satur datus par pieņemtajiem lēmuma veidiem – Izmaksa, Atteikums, Norēķinu korekcijas. Šis lauks ir obligāts LTAB datu bāzē līdz ar to nesatur null vērtības. Kopējais lēmuma veidu skaits ir 14, no kuriem lielākie ir “Izmaksa” un “Atteikums” (sk. 2.5. att. **Lēmuma veidu sadalījums**)



2.5. att. **Lēmuma veidu sadalījums**

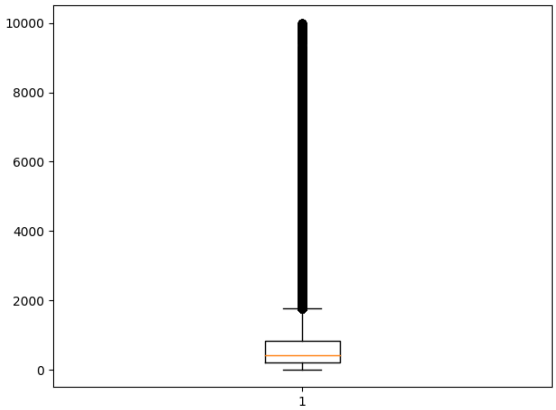
2.4.2.11. Atlīdzību summa

Lai varētu izmantot apdrošināšanas atlīdzību summas klasifikācijas uzdevumu risināšanā, tika izmantots atlīdzību sadalījuma dati un izveidots papildus lauks "SUM_INTERVALS".



2.6. att. **Atlīdzību summu histogrammas**

Aplūkojot atlīdzību izmaksu datus tika konstatēts, ka 7734 atlīdzībām izmaksu summa ir 0 Eur, kas nozīmē, ka vai nu tie ir lēmumi atteikt atlīdzību izmaksas vai arī datu kļūdas. 11861 atlīdzībām izmaksājamā atlīdzību summa nepārsniedza 20 Eur, kas nozīmē ka šie ieraksti satur kļūdainus datus, transakcijas par savstarpējiem norēķiniem starp apdrošināšanas sabiedrībām vai citu atlīdzību izmaksu summu korekcijas. Kopumā aplūkojot vairāk kā 546 tūkstošus ierakstu vidējā atlīdzību summa bija 1050 Eur, bet standart novirze 5014. Aplūkojot kvantiles, var redzēt, ka 75% datu apdrošināšanas atlīdzības nepārsniedz 10000 Eur, lai gan maksimālā summa sasniedz 2M Eur. No datu histogrammām (sk. 2.6. att. **Atlīdzību summu histogrammas**), gan uz normālās, gan logaritmiskās skalas var redzēt, ka atlīdzību summas lielākoties sakārtojas robežās līdz aptuveni 10000 Eur. Pārējos datus varētu uzskatīt par izlēcējiem. Tālāk lielāku uzmanību pievērsu tieši atlīdzībām līdz 10000 Eur .

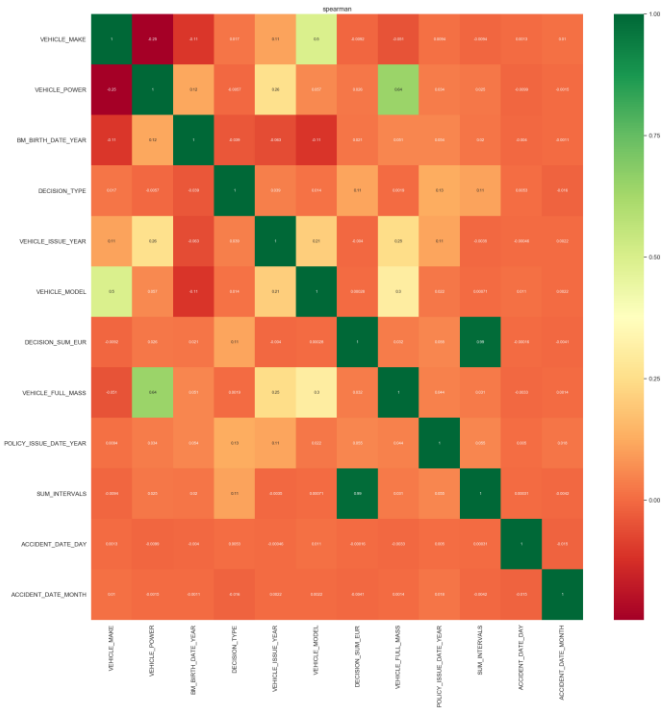


2.7. att. **Atlīdzību summu kastveida diagramma**

Skatoties atlīdzību summu kastveida diarammu (sk. 2.7. att. **Atlīdzību summu kastveida diagramma**) var redzēt, ka lielākā daļa atlīdzību ir sadalītas intervālā līdz 2000 Eur. Šī informācija ir būtiska, lai tālāk veidotu atlīdzību summu klases. Ir jāņem vērā arī tas, ka dati ir apkopoti sākot no 2008 gada un ka vieni no apdrošināšanas atlīdzību ietekmējošajiem faktoriem ir transportlīdzekļu rezerves daļu izmaksas un servisu darba stundu likmes. Šos faktorus ietekmē gan inflācija, gan kopējā ekonomiskā situācija, gan dzīves līmeņa izmaiņas. Šī darba mērķis nebija ekonomisko parametru analīze, līdz ar to, lai samazinātu ekonomisko faktoru ietekmi, apmācot modeļus, pētījums tika veikts gan ar visu datu kopu, gan ar atsevišķām datu kopām, kas tika sadalītas pa gadu intervāliem. Sīkāk tas tiks apskatīts turpmākā darba gaitā.

2.4.3. Datu korelāciju analīze

Sakarību analīze starp datiem ir būtiska, lai noteiktu kādas pazīmju kopas visvairāk ietekmē apdrošināšanas atlīdzību lielumu. Lai iegūtu vispārēju priekšstatu vai pastāv lineāras sakarības starp datiem tika izveidotas korelāciju matricu intensitātes kartes. (sk. 4. pielikums. Korelācijas koeficientu intensitātes kartes). Tika izmantoti divi korelāciju veidi [17] Introduction to Correlation [Tiešsaite]
https://www.datascience.com/learn-data-science/fundamentals/introduction-to-correlation-python-data-science
[18] Tensorflow Premade Estimators
https://www.tensorflow.org/guide/premade_estimators– Pīrsona korelācijas koeficients un Spīrmana ranka korelācijas koeficients. Pīrsona korelāciju koeficients parāda lineārās sakarības starp datiem, savukārt Spīrmana ranka koeficients rāda vai ir monotonas sakarības starp datiem/ Pētot datu kopas par dažādiem laika intervāliem, nekādas būtiskas lineāras sakarības starp datiem netika novērotas. Lielākā lineārā ietekme uz datiem bija lēmuma veidam, bet tas ir loģiski izskaidrojams ar to, ka lēmumiem ar tipu "Izmaksa" ir lielāka izmaksājamo summu vērtības nekā pārējiem tipiem.

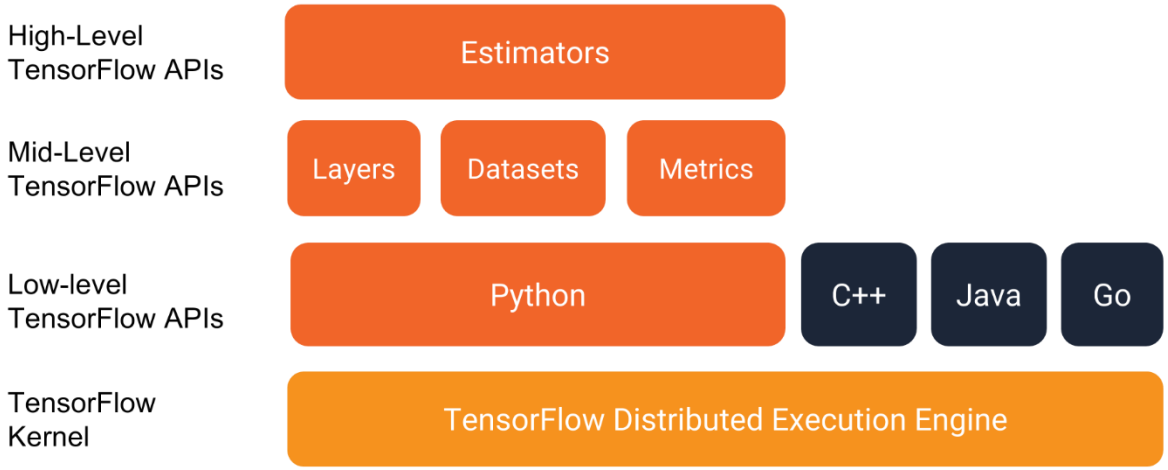


2.8. att. **Korelāciju intensitātes karte**

3. ESTIMATORU IZVEIDE IZMANTOJOT TENSORFLOW IETVARU

Tensorflow ir atvērtā koda programmatūra, ar kuras palīdzību var izstrādāt un attīstīt dažādus mašīnmācīšanās modeļus, tai skaitā arī dziļos neironu tīklus. Tensorflow ietvarā ir izveidotas lietotājam draudzīgas mašīnmācīšanās bibliotēkas, piemēram TfLearn [18] Tensorflow Premade Estimators
https://www.tensorflow.org/guide/premade_estimators

un Keras. Tensorflow ietvars nodrošina dažādu arhitektūru izmantošanu, lai apmācītu un lietotu dziļās mašīnmācīšanās modeļus. Tensorflow ļauj izmantot gan CPU, gan GPU aprēķinus, kā arī lietot to uz mobilajām ierīcēm. Tensorflow API versija 1.13 atbalsta tādas programmēšanas valodas, kā Python, C++, Java, Go, u.c., bet tikai Python tiek nodrošināta izstrādātāju API stabilitātes garantijas. Tensorflow API sastāv no vairākiem līmeņiem – augstākie līmeņi ir ar lielāku abstrakcijas līmeni, līdz ar to ir vieglāk lietojami, bet mazāk konfigurējami.



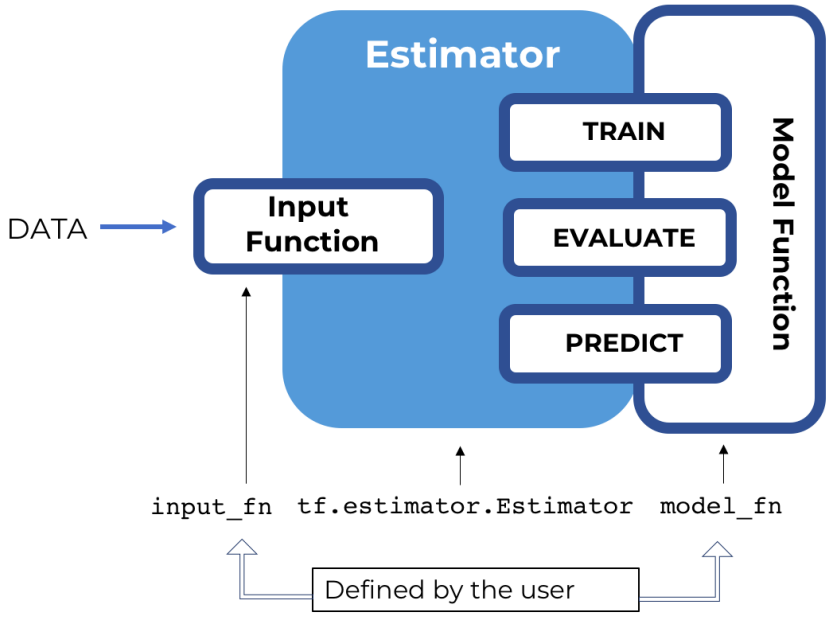
3.1. att. **Tensorflow API arhitektūra [19]**

Šajā darbā tiek lietoti dažādi API līmeņi, bet lielākais uzsvars tiek likts uz Tensorflow Estimatoru pakotni tf.estimator.*. Šīs pakotnes izveides mērķis bija izveidot viegli konfigurējamus un izvietojamus mašīnmācīšanās modeļus, kurus var izpildīt izmantojot dažādas arhitektūras aparatūru un vides. Galvenās estimatoru pakotnes priekšrocības ir:

- Iespēja izpildīt gan uz lokālām darbstacijām vai serveriem, gan attālinātos un sadalītos datu centros
- Iespēja izpildīt modeļus izmantojot CPU, GPU un TPU arhitektūras
- atvieglo iespēju vairākiem modeļa izstrādātājiem strādāt pie vienas implementācijas
- ļauj rakstīt intuitīvi saprotamu programmatūras pirmkodu
- modeļa izveides posmi ir secīgi un strukturēti (grafa, izveide, mainīgo inicializēšana, datu ielāde, kļūdu apstrāde, u.t.t.)

3.1. Estimatoru pakotnes arhitektūra

Tensorflow pakotne tf.estimator.Estimator jau satur izstrādātus gan klasifikācijas gan regresijas modeļus. Uz šo modeļu bāzes var ļoti ātri veidot un konfigurēt savus modeļus. Šīs estimatoru pakotnes īpašības tiek izmantotas šajā darbā un šikāk tiks aprakstīts nākamajās nodaļās. Estimatoru interfeiss satur trīs pamat metodes: apmācība – train, testēšana/precizitātes novērtēšana – evaluate un pārbaude/reāla darbība/prognozēšana – predict.



3.2. att. **Estimatoru objekta uzbūve [19]**

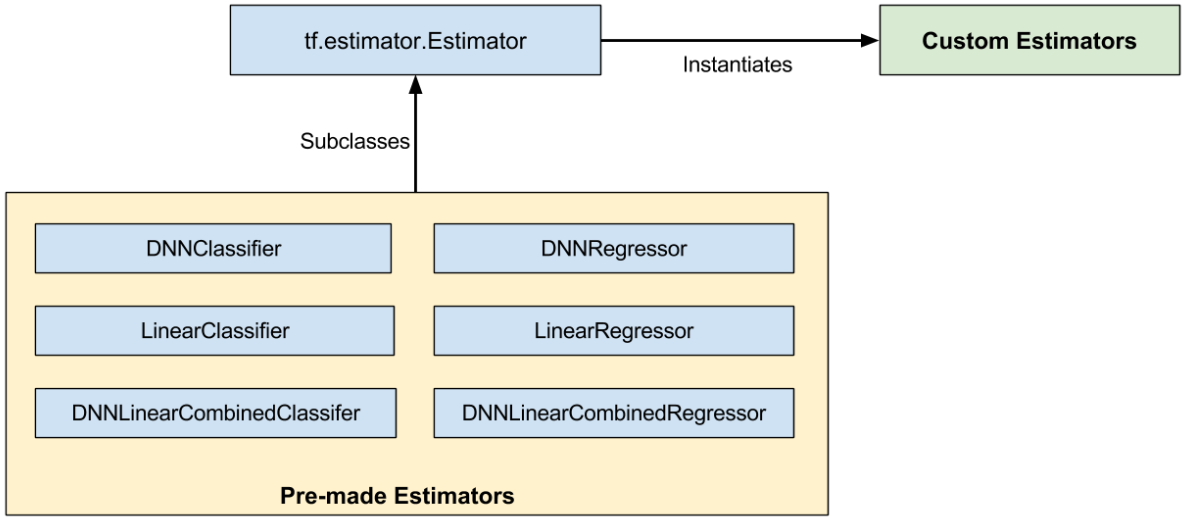
Estimatoru pakotne satur gan lineāros gan dziļās mašīnmācīšanās modeļus, bet atkarība modeļa veida tam var izmantot jau gatavas parametru vai metožu kopas, lai konfigurētu modeli. Piemēram:

- feature_columns - datu pazīmju lauku saraksts
- hidden_units - neironu tīkla slēpto līmeņu skaits un neironu skaits tajos, piemēram [128,64,16] norāda, ka tīkls sastāv no 3 neironu slāņiem, kur pirmajā ir 128 neironu, otrajā ir 64 neironi un trešajā 16 neironi
- optimizer – optimizātora veids, piemēram Adagrad, Adam, vai cits
- activation_fn - aktivācijas funkcija, kas tiks pielietota katrā līmenī, piemēram relu
- input_fn - datu kopas gan pazīmēm gan rezultāta laukiem

Veicot modeļa apmācību, visa modeļa informācija ik pa laikam tiek saglabāta, līdz ar to ir iespējams to ielādēt atkārtoti neveicot visa modeļa pārapmācību.

3.2. Iebūvētie estimatori

Tensorflow ietvars satur jau iepriekš izveidotu estimatoru objektus - trīs klasifikatoru estimatorus un trīs estimatorus regresiju veikšanai DNN Classifier, Linear Classifier, DNN Linear Combined Classifier, DNN Regressor, Linear Regressor, DNN Linear Combined Regressor.



3.3. att. **Estimatoru klases[19]**

DNNCalssifier un DNNRegressor ir predefinēts dziļo neironu tīkla modelis, kuram var veikt konfigurāciju norādot optimizātoru, aktivācijas funkcijas, slēpto līmeņu un neironu skaitu. No implementācijas viedokļa abi šie modeļi ir gandrīz identiski, klasifikatora modulis mēģina noteikt jeb prognozēt diskrētas vērtības jeb klases, bet regresora objekts prognozē absolūtas nepārtrauktas vērtības. Lineārais klasifikators un lineārais regresors izmanto klasiskās mašīnmācīšanās metodes, lai risinātu regresijas un klasifikācijas uzdevumus. DNN lineārie kombinētie modeļi ietver abas metodes vienā metodē, tas nozīmē, ka dažas pazīmes izmanto lineārās metodes bet citas dziļos neironu tīklus.

3.3. Pazīmju kolonas

3.3.1. Skaitļu pazīmju kolona

Tensorflow ietvara estimatoru pakotnē ir izveidotas vairāki pazīmju kolonu veidi. Pamat kolonas veids ir Skaitļu tipa kolona tf.feature_column.numeric_column. Šis kolonas veids ir paredzēts, lai ietvertu modelī skaitliskās datu lauku vērtības, ja nav nepieciešams kā savādāk tās grupēt vai sadalīt kategorijās. Pēc noklusējuma šī tipa kolona satur vienu vērtību nevis vektorus. Šajā darbā šī tip kolonas tika izmantotas, lai apstrādātu tādus datu laukus, kā datumu gada, mēneša un dienas informāciju.

3.3.2. Intervālu pazīmju kolona

Ir situācijas, kad ir nepieciešams apskatīt nevis karu skaitlisko vērtību atsevišķi bet gan sagrupētā veidā. Intervāla pazīmju kolonu veids ir paredzēts, lai cipariska satura datus sadalītu "grozos", balstoties uz skaitļu intervāliem. Piemēram, šajā darbā kā pazīmju kolona tika izmantota transportlīdzekļa jaudas dati. Datu lauks - VEHICLE_POWER. Datu vērtības ir robežās no 2 – 3301. Katrai jaudas vērtībai atbilst ne pārāk liels piemēru skaits, it sevišķi ja ņem vērā ka datu kopa tiek dalīta 3 daļās (treniņa, testa un pārbaudes).

3.4. att. **Transportlīdzekļa jaudas sadalījums grozos**

Šīs kolonas informācija tiek attēlota kā vektors un šajā piemērā tas sastāvēs no 5 elementiem:

- Grozs I [1,0,0,0,0]
- Grozs II [0,1,0,0,0]
- Grozs III [0,0,1,0,0]
- Grozs IV [0,0,0,1,0]
- Grozs V [0,0,0,0,1]

Šāda veidā modelim ir iespēja iemācīties ne tikai vienu vērtību, bet 5 vērtības uzreiz un tas ļauj modelī ietvert ne tikai lineāras sakarības.

3.3.3. Kategoriju identitātes pazīmju kolona

Kategoriju identitātes pazīmju kolona tf.feature_column.categorical_column_with_identity ir speciāls intervāla pazīmju kolonas veids, kurā katrs intervāls tiek attēlots kā unikāls vesels skaitlis. Piemēram, ja apskatām personas dzimumu datu lauku "BM_GENDER", tas sastāv no 3 vērtībām: "S", "V", "(n/z)", tad šīs vērtības var aizvietot ar skaitļiem 1 -"S", 2 - "V" un 3 - "(n/z)" . Tālāk jau vērtības tiks izmantotas līdzīgi intervāla pazīmju kolonām, kur vērtības tiek kodētas izmantojot "one-hot" kodējumu:

- 1 -"S" [1,0,0]
- 2 - "V" [0,1,0]
- 3 -"(n/z)" [0,0,1]

3.3.4. Kategoriju vārdnīcas pazīmju kolona

Kategoriju vārdnīcas pazīmju kolona ir līdzīga kā identitātes pazīmju kolona. Vienīgā atšķirība ir, ka kategoriju informācija tiek kodēta automātiski, izmantojot vārdnīcu. Vārdnīcu var norādīt divos veidos:

- kā sarakstu - tf.feature_column.categorical_column_with_vocabulary_list
- kā failu - tf.feature_column.categorical_column_with_vocabulary_file

Piemēram:

```
gender_feature_column = tf.feature_column.categorical_column_with_vocabulary_list(
    key=feature_name_from_input_fn, vocabulary_list=["S", "V", "(n/z)"])
```

3.3.5. Hash pazīmju kolona

Ja dati satur pārāk daudz kategoriju, tad var gadīties, ka pastāv tehniski ierobežojumi, piemēram atmiņas, tāpēc var izmantot hash pazīmju kolonu tf.feature_column.categorical_column_with_hash_bucket. Katrai datu klasei tiek sarēķināta hash atslēga un, ja intervālu skaits ir mazāks nekā vienādo hash atslēgu skaits, tad vairākas kategorijas tiek ievietotas vienā intervālā. Tādā veidā ir iespējams pazaudēt kategoriju informāciju, bet tas ļauj ietaupīt ka nav īpaši jāpēta visas iespējamās datu klases. Tas ir īpaši svarīgi situācijās, kad dati modeļa apmācības laikā papildinās un arvien veidojas jaunas datu klases. Šajā darbā tas tiek izmantos transportlīdzekļu marku un modeļu ievietošanai modelī.

3.3.6. Kombinētā pazīmju kolona

Kombinētā pazīmju kolona tf.feature_column.crossed_column ļauj savietot vairākas datu pazīmes vienā. Šajā darbā šis kolonas veids netiek izmantots, bet to varētu lietot lai apvienotu vienā pazīmē, piemēram transportlīdzekļa marku un modeli.

4. APDROŠINĀŠANAS ATLĪDZĪBU SUMMU UN REZERVJU PROGNOZĒŠANA

Lai varētu veikt korektu apdrošināmo risku izvērtēšanu un klientam piedāvāt maksimāli precīzu un personalizētu apdrošināšanas prēmijas piedāvājumu, kā arī nodrošinātu uzņēmuma maksātspēju, uzkrājot rezerves un būtiski ir pietiekami precīzi prognozēt iespējamā zaudējuma iestāšanās iespējamību kā arī novērtēt tā apjomu. Šajā nodaļā ir aprakstīts kā šis uzdevums tika veikts izmantojot gan klasiskās mašīnmācīšanās metodes, gan dziļos neironu tīklus. Šajā darbā es risināju vairākus uzdevumus.

Pirmais bija apskatot visu datu kopu, atbilstoši izvēlētajām apdrošināšanas objekta (transportlīdzeklis/persona) pazīmēm prognozēt apdrošināšanas atlīdzības lielumu vai arī noteikt apdrošināšanas atlīdzības intervālu, kurā ar vislielāko varbūtību varētu atrasties apdrošināšanas gadījuma atlīdzība citam objektam.

Otrais bija apskatot datus par iepriekšējiem gadiem, noteikt kādas atlīdzības varētu būt nākamajos gados.

Trešais uzdevums bija prognozēt apdrošināšanās atlīdzības lēmumu (izmaksa, atteikums, regress, u.t.t.), izmantojot apdrošināšanas objekta, negadījuma parametru un atlīdzības summas datus.

4.1. Izstrādes vide

Izstrādājot risinājumu tiks izmantotas vairākas Tensorflow konfigurācijas, kuras tiks ievietotas Docker konteinerī, kā arī lokālā instalācija, jo GPU konfigurācija docker konteinerī windows operētājsistēmā netiek atbalstīta. Docker konfigurācija tiks izmantota modeļa programmēšanas procesā, bet lokālā konfigurācija modeļa apmācībai.

4.1.1. Docker konteinera izveide

CPU

1) Instalē DockerToolbox https://www.docker.com/products/docker-toolbox
2) Izveido Dockerfile, kur raksta konteinera izveidei vajadzīgās darbības
3) Izveido konteineri docker build -t tensorflow-cpu-1.4 -f D:/work/Tensorflow/Dockerfile D:/work/Tensorflow
4) Ievieto konteineri publiskā repozitorijā, lai varētu piekļūt no citām darbstacijām
   docker login
   docker tag tensorflow-cpu-1.4 imapet/repo:tensorflow-cpu-1.4
   docker push imapet/repo:tensorflow-cpu-1.4
5) Palaist docker

docker login (gadījumos, ja repozitorijs nav publisks)

docker run -it -p 8888:8888 imapet/repo:tensorflow-cpu-1.4

GPU

Lai izmantotu docker ar GPU atbalstu, jāizmanto nvidia-docker utilīta, bet tā uz Windows platformas nav pieejama.

4.1.2. Lokāla instalācija

1) Instalē CUDA 8, jo Tensorflow 1.4 neatbalsta jaunākas CUDA versijas https://developer.nvidia.com/cuda-80-ga2-download-archive
2) Instalē CuDNN 6.0 https://developer.nvidia.com/rdp/cudnn-download (Jāreģistrējas NVIDIA)
3) Papildina PATH vides mainīgo ar bibliotēkas atrašanās vietu
4) Instalē Python 3.6 https://www.python.org/downloads/release/python-363/
5) Papildina PATH vides mainīgo ar C:\Program Files\Python36\Scripts
6) Instalē Tensorflow
   pip3 install --upgrade tensorflow
   pip3 install --upgrade tensorflow-gpu
7) Pārbauda vai instalācija ir pareiza

4.1. tabula

Python programma	Rezultāts
import tensorflow **as** tf hello = tf.constant(**'Hello, TensorFlow!'**) sess = tf.Session() print(sess.run(hello)))	2017-11-19 16:57:46.454781: I C:\tf_jenkins\windows-gpu\PY\36\tensorflow\core\common_runtime\gpu\gpu_device.cc:1030] Fonus device 0 with properties: name: GeForce GTX 1080 Ti major: 6 minor: 1 memoryClockRate(GHz): 1.683 pciBusID: 0000:01:00.0 totalMemory: 11.00GiB freeMemory: 9.10GiB 2017-11-19 16:57:46.455037: I C:\tf_jenkins\windows-gpu\PY\36\tensorflow\core\common_runtime \gpu\gpu_device.cc:1120] Creating TensorFlow device (/device:GPU:0) -> (device: 0, name: GeForce GTX 1080 Ti, pci bus id: 0000:01:00.0, compute capability: 6.1) b'Hello, TensorFlow!'

4.2. Rezervju aprēķina un atlīdzību datu prognozēšanas modeļu konfigurācijas programmatūra

Darba ietvaros uz Tensorflow pamata tika izveidots risinājums ar kura palīdzību var veikt dažādu arhitektūru tīklu apmācību, ir viegli veikt modeļu parametrizāciju visiem tīklu veidiem, kā arī hiperparametru optimizāciju. Ir iespēja arī iegūtos rezultātus ierakstīt failos, lai vēlāk būtu ērti tos salīdzināt. Programmatūras pirmkodu skatīt 10. pielikums. Rezervju aprēķina modeļu un konfigurācijas pirmkods

4.2.1. Konfigurācija

Modulis Configurations.py ir atbildīgs par modeļa konfigurācijas ievadi. Šajā modulī tiek definētas piedefinētas konfigurācijas.

Klase Configuration satur vienu konfigurācijas ierakstu. Funkcija get_config(index = 0) atgriež vienu konfigurācijas objektu pēc norādītā indeksa.

4.2.2. Datu ielāde un pirmsapstrāde

Modulis InsuranceDataset.py nodrošina datu ielādi no teksta failiem un to sagatavošanu ielādei mašīnmācīšanās modeļos. Funkcija load_dataset_from_file ielādē datus no norādītā faila un izveido datu kolonu nosaukumus. Funkcija clean_and_fix_data veic datu labošanu, nederīgo ierakstu dzēšanu, datu tipu konvertāciju un nevajadzīgo kolonu izmešanu. Funkcija get_datasets_with_labels atgriež testa, treniņa un validācijas testa kopas. Katru kopu veido kortežs, kur pirmā vērtība ir dati, bet otra vērtība ir meklējamās/ mērķa vērtības (labels). Ar parametriem var norādīt cik procentuāli lielas veidot šis kopas.

4.2.3. Modeļa izveide un konfigurēšana

Modulis ModelFactory.py nodrošina sistēmas parametru iestatīšanu, modeļu izveidi, konfigurēšanu un izpildi. Funkcija create_model nodrošina modeļa izveidi atbilstoši uzdotajam konfigurācijas parametram. Funkcija run_model nodrošina parametrā norādītā modeļa izpildi, rezultātu apkopošanu un izvadi teksta failā. Funkcija create_features_columns nodrošina pazīmju kolonu izveidi. Izveide tiek veikta atbilstoši nodotajam pazīmju kolonu nosaukuma parametram, kurā norādīts, kuri datu lauki būs pazīmju kolonas modelī. Funkcija print_field_statistics nodrošina parametrā norādītā datu lauka statistiku, histogrammu un kastu diagrammas izvadi uz ekrāna. Funkcija calculate_features_importance veic izvēlēto pazīmju kolonu svarīguma izvērtējumu ar Pīrsona un Spīrmana korelāciju matricām un hī2 testu palīdzību.

4.2.4. Mašīnmācīšanās modeļi

Risinājumā ir ietverti 6 konfigurējami mašīnmācīšanās modeļi. Divos no tiem ir implementēti lineāri klasifikācijas un regresijas modeļi – LinearRegressor.py un LinearClassifier.py. Pārējos modeļos ir izmantoti dziļo neironu tīklu arhitektūra. DNNRegressor.py satur dziļo neironu tīklu regresijas modeli, DNNClassifier.py - dziļo neironu tīklu klasifikācijas modelis. Lai varētu salīdzināt pēc iespējas dažādāku tīklu arhitektūru, tika izveidots arī dziļo rekurento tīklu modelis klasifikācijas uzdevumu veikšanai RNNClassifier.py. Visi modeļi satur vienu funkciju run_model, kas satur gandrīz identiskus parametrus un nodrošina modeļa apmācību, testēšanu un validāciju, kā arī izpildes datu vākšanu tālākai analīzei.

4.2. tabula

run_model funkcijas parametri

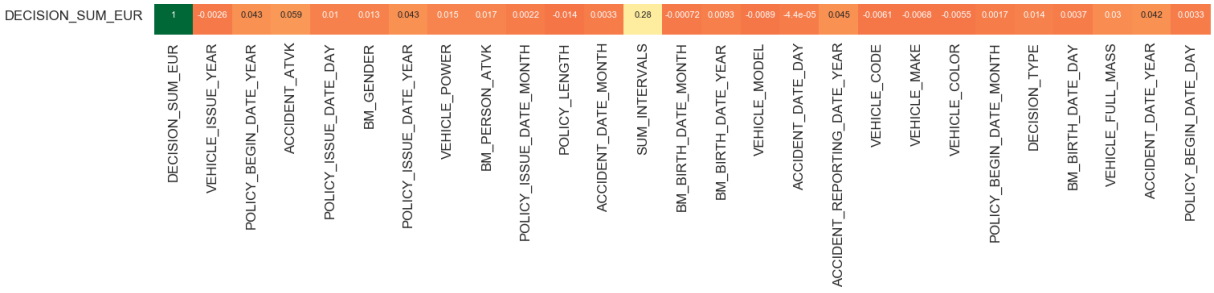
Parametrs					
	DNN classifier	**DNN regressor**	**RNN classifier**	**Linear classifier**	**Linear regressor**
feature_columns	X	X	X	X	X
train_dataset	X	X	X	X	X
test_dataset	X	X	X	X	X
model_dir	X	X	X	X	X
hidden_units	X	X	X	-	-
optimizer	X	X	X	X	X
activation_fn	X	X	-	-	-
dropout	X	X	-	-	-
batch_size	X	X	X	X	X
num_epochs	X	X	X	X	X
steps	X	X	X	X	X
label_vocabulary	X	-	-	X	-
norm_factor	-	X	X	-	X

4.3. Pazīmju kolonu izvēle un konfigurācija

Pazīmju izvēle ir viena no būtiskākajām veicamajām darbībām, kas nozīmīgi var ietekmēt modeļa darbības veiktspēju. Nesvarīgu pazīmju iekļaušana modelī būtiski samazina modeļa precizitāti. Pareiza pazīmju izvēle ļauj samazināt trokšņu līmeni datos un samazina modeļa pārpielāgojamību, palielina modeļa precizitāti un uzlabo apmācības laiku. Jāņem vērā, ka datu kopā tika iekļauti dati par 10 gadu lielu periodu un sājā laikā gan ekonomiskie, gan sociālie apstākļi, gan apdrošināšanas sabiedrību tehniskie risinājumi un ar to saistītie datu kvalitātes jautājumi atstāja iespaidu gan uz apdrošināšanas industrijas attīstību, gan zaudējumu regulēšanas procesu un apmēru. Tāpēc lai mazinātu šo apstākļu ietekmi, testi tika veikti gan uz visu datu kopu gan sadalot datus pa daļām. Pirmais kritērijs bija sadalīt datus pēc apdrošināšanas līguma izdošanas gada - ņemot ierakstus jaunākus par 2013 gadu, pēc tam jaunākus par 2015 gadu un 2018 gadu un jaunākus. Otrs kritērijs bija lēmumu pieņemšanas veids. Lielāko īpatsvaru sastāda dati ar status “Izmaksa” un ”Atteikums”. Trešais kritērijs bija transportlīdzekļa veids. Tā kā dati tiek saņemti no divām iestādēm CSDD - par vieglajiem un kravas transportlīdzekļiem un VTUA - par traktortehniku un lauksaimniecības tehniku. Izdarīju uz pieredzi balstītu pieņēmumu, ka VTUA dati satur mazāk precīzu informāciju, tad tika izmantota datu kopa tikai par vieglajiem transportlīdzekļiem. Otrs pieņēmums bija, ka jauni transportlīdzekļi garantijas laikā tiek remontēti dīleru servisos, kur servisa izmaksas ir lielākas nekā citos servisos. Līdz ar to tika izveidota datu kopa ar vieglajiem transportlīdzekļiem, kuri negadījuma brīdī nav vecāki par 3 gadiem.

Kopuma tika izveidotas 24 pazīmes, kuras aprakstītas zemāk redzamajā tabulā. Pirms veidot pazīmju kolonas tika veikta datu analīze lai saprastu cik daudz grupas jeb klases, katrs lauks satur. Pazīmju kolonu aprakstus skatīt.

Šajā darbā tika izmantotas trīs metodes, lai noskaidrotu perspektīvākās pazīmes, kuras varētu lietot modeļa apmācībā. Pirmā metode, kas tika izmantota bija Pīrsona korelāciju matrica, lai noteiktu cik stipras lineāras sakarības pastāv starp datiem, bet otra - Spīrmana ranku korelācijas, lai noskaidrotu vai pastāv monotonas sakarības starp datiem.



4.1. att. **Korelāciju intensitātes karte atlīdzību summas laukam**

Visas korelāciju matricu intensitātes kartes var redzēt 4. pielikums. Korelācijas koeficientu intensitātes kartes. Analizējot iegūtos rezultātus nevarēja novērot būtiskas sakarības starp Atlīdzību summu, lēmuma veidu un apdrošinātā objekta un personas datiem (sk. 4.1. att.).

Trešā metode, kura tika izmantota bija, pazīmju atlase izmantojot $Hī^2$ testu. $Hī^2$ testu izmanto statistikā, lai noteiktu divu notikumu neatkarību. Zemāk ir redzamas atrastās 10 labākās pazīmes.

4.3. tabula

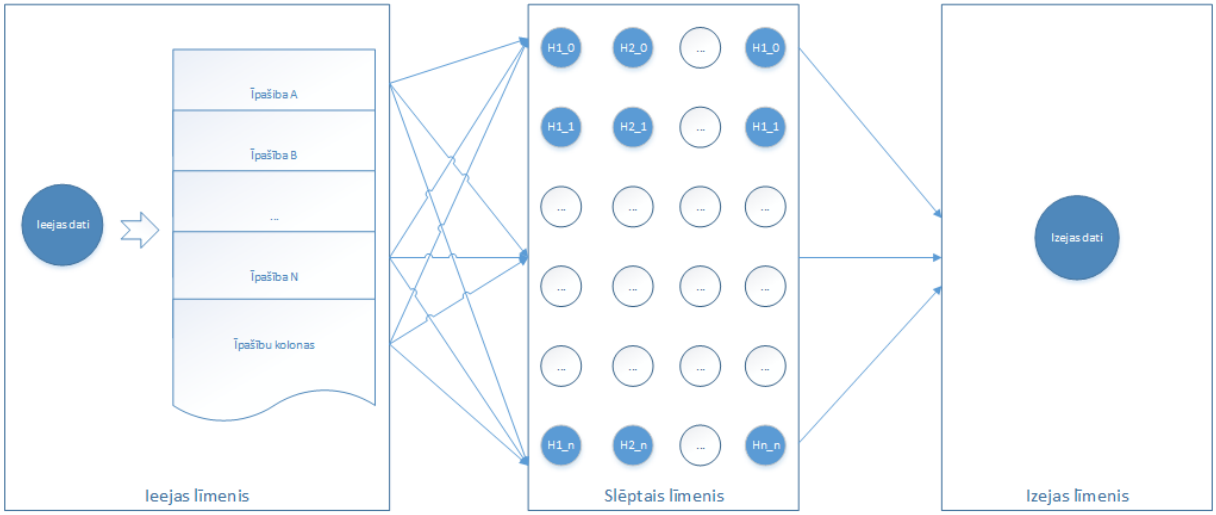
Pazīmju izvēles rezultāti

Pazīme	Rezultāts
ACCIDENT_ATVK	1.475935e+08
BM_PERSON_ATVK	2.077594e+07
VEHICLE_FULL_MASS	1.635025e+05
DECISION_TYPE	8.286063e+03
VEHICLE_MODEL	5.811626e+03
POLICY_LENGTH	1.999617e+03
VEHICLE_POWER	1.431514e+03
VEHICLE_COLOR	5.153533e+02
VEHICLE_MAKE	3.505691e+02
POLICY_ISSUE_DATE_DAY	9.554305e+01
BM_BIRTH_DATE_DAY	3.238225e+01
BM_GENDER	3.997003e+01
BM_BIRTH_DATE_MONTH	3.238225e+01

Tīkls tika apmācīts gan ar visām pazīmju kolonām gan top 7 pazīmēm. Iegūtos rezultātus var redzēt 6. pielikums. Neironu tīkla modeļu zaudējumu funkcijas rezultāti.

4.4. Neironu tīkla konfigurācija

Pētījuma laikā tika eksperimentēts ar dažāda platuma un dziļuma tīkliem, kā arī tika izmantoti dažādi optimizātora un izmantotas dažādas īpašību kolonu kombinācijas. Vispārējo neironu tīkla shēmu (sk. 4.2. att. **Neironu tīkls**)



4.2. att. **Neironu tīkls**

Neironu tīkla modeļa izveidei tika izmantota tensorflow pakotne tf.estimator un tajā ietilpstošās klases: DNNRegressor un DNNClassifier, un eksperimentālo klasi RNNClassifier. Šī klase piedāvā gatavu neironu tīkla implementāciju, kuru iespējams parametrizēt. Modeļu parametrus sk. 4.4. tabula.

4.4. tabula

Modeļu inicializācijas parametri

Parametra nosaukums	Parametra apraksts
hidden_units	Masīvs, kurā var norādīt katrā slēptajā līmenī esošo neironu skaitu, piemēram [64, 32, 8] – norāda, ka ir definēti 3 neironu līmeņi, kur pirmajā ir definēti 64, otrajā - 32, bet trešajā 8 neironi
feature_columns	Ieejas datu pazīmju kolonu dati
model_dir	Direktorija, kurā tiek glabāti ar modeļa izpildi saistītie dati - parametri, grafi, u.c.
label_dimension	Regresijas mērķa kolonnu skaits, pēc noklusējuma 1
weight_column	Pazīmju kolonu svari
optimizer	Izmantotais optimizātora, pēc noklusējuma tiek izmantots Adagrad optimizātora
activation_fn	Aktivizācijas funkcija, pēc noklusējuma tiek izmantota RELU
dropout	"Dropout" varbūtības vērtība

4.5. Modeļu pielietošana un rezultāti

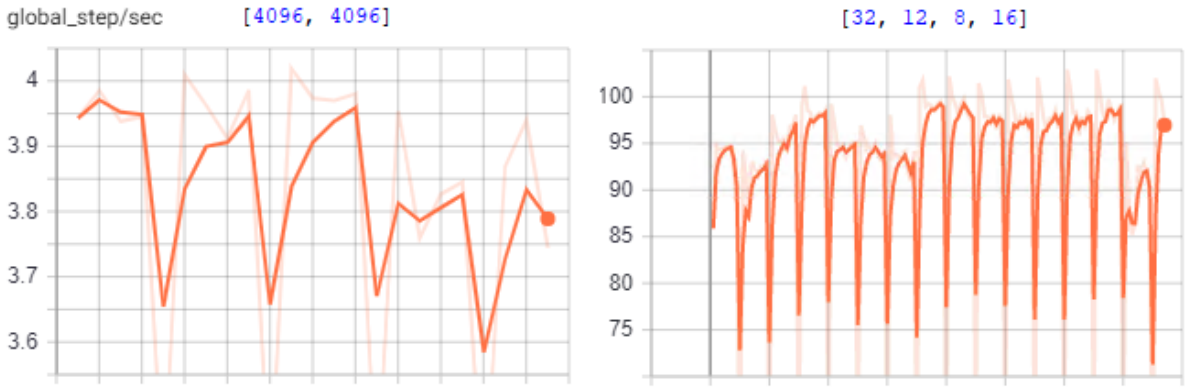
Šajā darbā tika apskatīti gan regresijas gan klasifikācijas uzdevumi, kuru mērķis bija noteikt iespējamo apdrošināšanas atlīdzību summu un summu intervālu nākamajiem periodiem. Atlīdzību rezerves tika noteiktas izveidojot un izmantojot dažādus neironu tīklu modeļus, pielietojot dažādus hiperparametrus, izvēloties vai nu visas vai top 7 pazīmju kolonu grupas. Veicot modeļa apmācību tika izmantotas randomizētas treniņa un testa datu kopas.

Tīklu apmācība tika sākta izmantojot Adam optimizātoru un relu aktivācijas funkciju, kā arī ar ļoti nelielu neironu skaitu vienā līmenī, izmantojot visas pazīmju kolonas. Sākotnēji apmācība tika veikta modelī ieslēdzot tikai vienu slēpto līmeni, izmantojot nelielu neironu skaitu. "dropout" vērtība tika norādīta 0.1 Pakāpeniski tika palielināts gan neironu skaits, gan slēpto līmeņu skaits. Palielinot līmeņu skaitu uz, zaudējumu funkcijas rezultāti uzlabojās, bet izmantojot tīklu dziļāku par 4 rezultāti vairāk neuzlabojās (sk. 4.5. tabula)

4.5. tabula

Slēpto līmeņu konfigurācija	Vidējā kļūda uz testa kopas (pēc 20000 soļiem)
[64]	0.225
[64, 32]	0.17
[128, 64, 32]	0.067
[32, 16, 2048, 16]	0.037
[32, 12, 8, 16]	0.024
[4096, 4096]	828.1
[4096, 2048, 2048, 16]	3227.5
[512, 256, 8]	0.0489
[512, 128, 128, 256]	0.0318
[32, 12, 8, 16, 12]	0.066

Tika novērots, ka liela neironu skaita izmantošana neuzlabo zaudējumu funkcijas rezultātu un modelis mācās būtiski lēnāk - pie viena un tā paša apmācības soļu skaita zaudējumu funkcijas rezultāts ir sliktāks. Izmantojot lielu neironu skaitu viens apmācības solis aizņēma krietni ilgāku laiku (sk. 4.3. att. **Apmācības soļa ilgums izmantojot dažāda lieluma tīklus**). Līdz ar to turpmākajā darba gaitā tika izmantots neironu tīkls, kurš satur 4 slēptos slāņus un neironu skaits nepārsniedz 64.



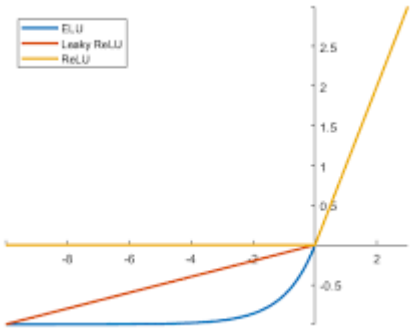
4.3. att. **Apmācības soļa ilgums izmantojot dažāda lieluma tīklus**

Mainot optimizātorus un mācīšanās soli, būtiskas izmaiņas modeļu darbības rezultā netika novērotas, bet visstabilākie rezultāti, kad zaudējuma funkcija konverģēja vismierīgāk un tajā pašā laikā netika zaudēts apmācības ātrums, tika iegūti izmantojot mācīšanās soli 0.01.

Tā kā pieejamās datu kopas izmērs bija ierobežots tika nolemts neizmantot pārāk lielu "batch" izmēru. Vislabākie rādītāji tika iegūti izmantojot "batch size" parametra vērtību 8. (sk. 8. pielikums. Zaudējumu noteikšanas klasifikācijas un regresijas rezultāti).

Mainot "dropout" vērtības, tam nebija būtiska ietekme uz gala rezultātu, bet tomēr bija novērojams neliels uzlabojums. Turpmākie izsaukumi tika veikti, izmantojot pilnu datu kopu, kas satur datus kopš 2008 gada. Iegūtie dati nebija iepriecinoši, jo gan regresijas, gan klasifikācijas rezultāti konverģēja uz datu kopas vidējām vērtībām, kas varētu liecināt par datu kopas nepietiekamību, kvalitātes problēmām. Rezultātu neietekmēja arī apmācības ilguma palielināšana un hiperparametru izmaiņas. (sk. 8. pielikums. Zaudējumu noteikšanas klasifikācijas un regresijas rezultāti).

Apskatot konverģences uz vērtībām problēmu tika veikti eksperimenti ar aktivācijas funkcijām [20]



4.4. att. **ELU, RELU, LEAKY RELU salīdzinājums**

Izmantojot leaky RELU funkciju konverģence uz vidējām vērtībām tika novērsta. To var izskaidrot ar to, ka RELU pieņem vērtības sākot no 0 un ja nozīmīga daļa gradientu ir negatīva, tas var novest, ka liela daļa neironu ir "miruši", tas ir modelis uzskata, ka šo neironu svaru vērtību maiņa rezultātu neuzlabos. Atšķirībā no RELU, leaky RELU ļauj izmantot arī nelielu negatīvu vērtību intervālu, kas augstāk minēto problēmu risina.

Salīdzinot lineāro un dziļo neironu tīkla modeli, varēja secināt, ka neironu tīkla modelis strādā nedaudz labāk(sk. 4.7. tabula), tomēr ar šo rezultātu jābūt uzmanīgiem, jo pētījuma laikā, neironu tīkla modeļa optimizācijai tika veltīta krietni lielāka uzmanība nekā lineārajam modelim. Uz visu datu kopu veiktie testi apliecināja, ka rezultātos nav būtiskas starpības starp lineāriem un dziļo neironu tīklu modeļiem.

4.6. tabula

DNN modeļa un Lineāra regresijas modeļa zaudējumu funkcijas rezultāta salīdzinājums

	Lineārs modelis	DNN modelis
Zaudējumu funkcijas rezultāti uz testa datu kopas	0.0391	0.0241
	0.0652	0.0311
	0.0401	0.0264
	0.0451	0.0244
	0.0397	0.0275

Salīdzinot abu modeļu mācīšanās ātrumu, jāpiemin, ka abi modeļi darbojās līdzīgi un abos gadījumos zaudējumu funkcijas rezultātus ieguva 1000 līdz 2000 soļu laikā un visā pārējā apmācības laikā rezultāti palika nemainīgi.

Veicot klasifikācijas uzdevumu tika salīdzināti divi neironu tīkla veidi - DNN modelis un rekurento dziļo neironu tīkla modelis un lineārais klasifikācijas. Veicot apmācību uz visiem datu kopas datiem tika iegūti līdzīgi rezultāti kā regresijas gadījumā.

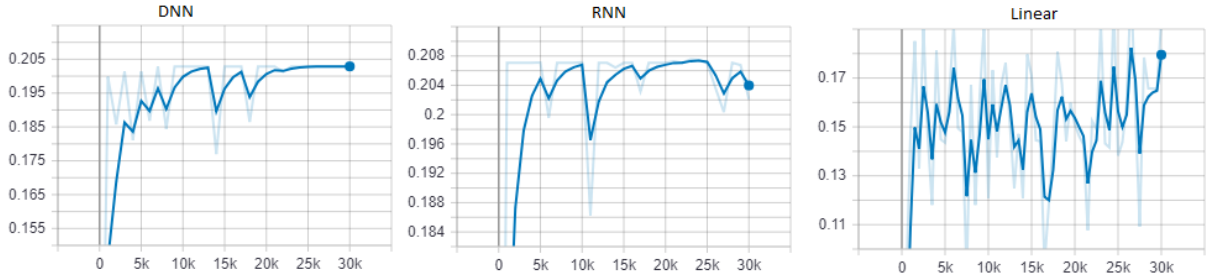
4.7. tabula

RNN un DNN klasifikācijas tīklu zaudējumu funkcijas rezultāta salīdzinājums

	RNN modelis	DNN modelis
Zaudējumu funkcijas rezultāti uz testa datu kopas	2.124	2.119
	2.396	2.438
	2.439	2.237
	2.401	2.279
	2.133	2.439

Būtiskas atšķirības apmācību rezultātos netika novērotas, bet var redzēt ka RNN tīkls strādā nedaudz labāk. Bet jebkurā gadījumā iegūtie rezultāti nebija pārliecinoši. Salīdzinot abu tīklu apmācības ilgumu, varēja novērt ka RNN tīkls izmantoja būtiski vairāk resursus un apmācība aizņēma vairākas reizes lielāku laiku. Piemēram, ja DNN tīklā ar 3 slēptajiem līmeņiem un nelielu neironu skaitu [128, 64, 16], 100 soļu apmācība aizņēma 0.8 sekundes, tad tādas pašas konfigurācijas RNN modelī tas aizņēma 0.4 sekundes. Palielinot slāņu skaitu un būtiski palielinot neironu skaitu uz [4096, 128, 64, 16], varēja novērot ka RNN modeļa apmācības laiks 100 soļiem kļuva vēl lēnāks - attiecīgi 11.2 pret 37.2 sekundēm.

Analizējot klasifikācijas rezultātus, tie bija slikti jo precizitāte bija tikai 20%. RNN un DNN tīkli darbojās ar līdzīgu precizitāti, bet lineārais modelis nedaudz sliktāk. (sk. 4.5 att. **Klasifikācijas precizitātes salīdzinājums**)



4.5 att. **Klasifikācijas precizitātes salīdzinājums**

No augstāk minētajiem rezultātiem tika secināts, ka lai iegūtu lietojamu rezultātu ir nepieciešams atbrīvoties no izlēcējiem datos kā arī izmest nesvarīgos ierakstus vai arī izņemt tās datu apakškopas, kas satur ļoti mazu ierakstu skaitu, piemēram specifiskus transportlīdzekļa modeļus, kā arī ierakstus, kas attiecas maz lietotiem atlīdzību lēmumu datiem. Tā kā datu kopa satur datus par daudziem gadiem un šie dati tika vēsturiski iegūti no apdrošināšanas sabiedrībām un uzkrāti izmantojot dažādus tehnoloģiskos risinājumus kā arī apdrošināšanas atlīdzību izmaksu līmenis un servisa pakalpojumu izcenojumi ir kāpuši, tādēļ tika izmantoti dati tikai par pēdējiem pāris gadiem. Vēl tika ņemts vērā apsvērums, ka transportlīdzekļi tiek remontēti gan autorizētos dīleru servisos, gan parastos servisos, kur izmaksu līmenis būtiski atšķiras, tāpēc tika pieņemts lēmums, ka tiks apskatīti transportlīdzekļi, kuru vecums negadījuma brīdī nepārsniedza 3 gadus. No datiem tika izņemti atlīdzību summu izlēcēji – apdrošinājumu atlīdzību summas, kas pārsniedz 10000 Eur. Rezultātā datu kopa tika būtiski samazināta.

Iegūtie rezultāti klasifikācijas un regresijas algoritmiem būtiski uzlabojās (sk. 7. pielikums. Modeļu treniņu un testu rezultāt).

5. SECINĀJUMI

Pēdējā laikā Tensorflow ietvara attīstība, dažādo API atbalsts un prekonfigurēto estimatoru izstrāde ir būtiski atvieglojusi dziļo neironu tīklu izmantošanu un integrēšanu biznesa aplikācijās.

Būtiska loma kvalitatīvu estimatoru izveidē ir datu apjomam un datu kvalitātei. Personas datu aizsardzības regulējuma ieviešana Eiropas Savienībā, būtiski samazina pieejamo datu apjomu, ko varētu uzkrāt un izmantot neironu tīklu modeļu apmācībai. Bieži vien ir nepieciešams izmantot konkrētā industrijā pieejamos datus, šādos gadījumos izmantot tikai Latvijas apdrošināšanas tirgū pieejamos datus ir nepietiekami, jo tad ir jāizmanto dati, kas ir iegūti daudzu gadu garumā un ir grūti nodrošināt to viendabīgumu. Starptautiskām korporācijām vai valstīm ar lielu biznesa transakciju skaitu ir priekšrocības, jo ir iespēja iegūt krietni lielāka apjoma datus, bet neskatoties uz to tāpat ir jāņem vērā tirgus specifiskie apstākļi.

Liela nozīme datu apstrādē un analīzē ir izlēcējiem. Risinot regresijas uzdevumus dziļie neironu tīkli konverģē uz šim vērtībām. Samazinot izlēcēju ietekmi un dispersijas datos izdevās būtiski uzlabot modeļu veiktspēju.

Izmantoto pazīmju skaits modeļu precizitāti būtiski neietekmēja. Zaudējumu funkcija atgrieza līdzīgus rezultātus gan izmantojot visas pazīmju kolonas, gan tikai svarīgākās kolonas. Bet nebūtisko pazīmju atmešana tomēr var būt būtiska, lai samazinātu modeļa apmācības laiku.

Izveidotajiem modeļiem zaudējumu funkcija dilst gan izmantojot treniņa, gan testa datu kopām, kas norāda, ka izveidotie modeļi darbojās. Izmantojot relu, elu, selu, sigmoid, tanh aktivācijas funkcijas un dažādus modeļa parametrus, zaudējumu funkcijas rezultāti būtiski nemainījās, bet iegūtās prognozes konverģēja uz datu kopas vidējām vērtībām, kas norāda, ka datu kopas apmērs bija nepietiekams. Eksperimentējot ar aktivācijas funkcijām konverģenci uz datu kopas vidējām vērtībām izdevās novērst izmantojot tikai leaky relu funkciju.

Tādu modeļa konfigurācijas parametru maiņa, kā slāņu skaits, neironu skaits slēptajos slāņos, hiperparametru optimizācija neatstāja būtisku ietekmi uz kopējiem modeļu darbības rezultātiem. Vislabākos rezultātus izdevās iegūt izmantot tīklu ar 4 slēptajiem slāņiem, kur neironu skaits ir mazāks par 100.

Slēpto slāņu un neironu skaita palielināšana tajos būtiski ietekmē apmācību ilgumu, bet modeļa veiktspēja prognozējot apdrošināšanas atlīdzību datus neuzlabojās. Sevišķi liels apmācību ilguma un izmantoto dator resursu pieaugums bija vērojams rekurentajiem neironu tīkliem. Lai gan rezultāts bija tikai nedaudz labāks, tomēr ir vērts turpināt pētījumus kā optimizēt rekurento tīklu darbību prognozēšanas uzdevumu veikšanai.

IZMANTOTĀ LITERATŪRA UN AVOTI

[1] AccentureConsulting [Tiešsaite] https://www.accenture.com/hk-en/insight-insurance-technology-vision-2017

[2] Munich RE, Investor Day 2017 [Tiešsaite] https://www.munichre.com/site/corporate/get/documents_E-1557060415/mr/assetpool.shared/Documents/0_Corporate_Website/5_Investor_Relations/Publications/Presentations/MunichRe-IDay2017-Presentation.pdf

[3] LTAB mājas lapa [Tiešsaite] https://www.ltab.lv/

[4] OCTA likums [Tiešsaite] https://likumi.lv/doc.php?id=87547

[5] Accenture innovation blog [Tiešsaite] https://insuranceblog.accenture.com/the-virtual-insurance-advisor

[6] A. Abdallah,M.A.Maarof, A. Zainal, Fraud detection system: a survey, Journal of Network and Computer Applications 68 (2016) 90–113. https://datubazes.lanet.lv:2076/science/article/pii/S1084804516300571

[7] E.W.T. Ngai, Y. Hu, Y.H.Wong, Y. Chen, X. Sun, The application of data mining techniques in financial fraud detection: a classification framework and an academic review of literature, Decision Support Systems 50 (3) (2011) 559–569. https://datubazes.lanet.lv:2076/science/article/pii/S0167923610001302

[8] Vipula Rawte, G Anuradha 2015 International Conference on Communication, Information & Computing Technology (ICCICT), Jan. 16-17 https://www.researchgate.net/publication/282538462_Fraud_detection_in_health_insurance_using_data_mining_techniques

[9] FRISS [Tiešsaite] https://knowledge.friss.com/survey-insurance-fraud-2019 https://www.friss.com/resources/

[10] Y. Wang, W. Xu, Leveraging deep learning with LDA-based text analytics to detect automobile insurance fraud, Decision Support Systems 105 (2018) 87–95 https://datubazes.lanet.lv:2076/science/article/pii/S0167923617302130

[11] Ibiwoye, A., Ajibola, O. O. E. and Sogunro, A. B. 2012. Artificial Neural Network Model for Predicting Insurance Insolvency, International Journal of Management and Business Research, 2(1) 59- 68. http://ijmbr.srbiau.ac.ir/article_63_573728fab7c0fc01238a2a8a5c4c9a22.pdf

[12] Mario V. Wuthrich and Christoph Buser 2016. " Data Analytics for Non-Life Insurance Pricing ", Swiss Finance Institute Research Paper No. 16-68
https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2870308
[13] Gelman, A. and J. Hill. 2007. Data analysis using regression and multilevelhierarchical models. Cambridge University Press New York, NY, USA.
https://faculty.psau.edu.sa/filedownload/doc-12-pdf-a1997d0d31f84d13c1cdc44ac39a8f2c-original.pdf
[14] Mario V. Wüthrich, Michael Merz Stochastic Claims Reserving Manual: Advances in Dynamic Modeling, 2015
http://www.aktuar.sk/userfiles/Wuthrich%20-%20book.pdf
[15] Kevin Kuoa, DeepTriangle: A Deep Learning Approach to Loss Reserving, 2019
https://arxiv.org/pdf/1804.09253.pdf
[16] Data Masking Best Pratices, Oracle White Paper, 2013
http://www.oracle.com/us/products/database/data-masking-best-practices-161213.pdf
[17] Introduction to Correlation [Tiešsaite]
https://www.datascience.com/learn-data-science/fundamentals/introduction-to-correlation-python-data-science
[18] Tensorflow Premade Estimators
https://www.tensorflow.org/guide/premade_estimators
[19] Tensorflow Feature Columns
https://www.tensorflow.org/guide/feature_columns
[20] Comparison of non-linear activation functions for deep neural networks on MNIST classification task
https://arxiv.org/pdf/1804.02763.pdf

PIELIKUMI

1. pielikums. Datu noliktavas dimensiju un faktu tabulu apkopojums

Filtri

Nosaukums	Apraksts
Laikaposms no - līdz	Dimensiju un faktu tabulās minēto notikumu datumu intervāli. Tiek izmantots datu filtrēšanai no datu noliktavas

Dimensijas

Nosaukums	Apraksts
Līgums	Apdrošināšanas līguma dati
Apdrošinātājs	Apdrošinātāja, kurš noslēdzis līgumu vai regulējis zaudējumus, dati
Persona	Apdrošināšanas procesā iesasitīto personu datu. Transportlīdzekļa īpašnieks, turētājs vadītājs, BM subjekts
Transportlīdzeklis	Transportlīdzeļu dati
CSNg	Dati par ceļu satiksmes negadījumu
Zaudējumu pieteikums	Dati apdrošināšanas zaudējumu pieteikumu
Lēmums	Dati par lēmumiem, kas pieņemti apr apdrošināšanas atlīdzības izmaksu
Maksājums	Dati par maksājumime, kas veikti par apdrošināšanas zaudējumu pieteikumu

Fakti

Nosaukums	Apraksts
CSNg	Dati par ceļu satiksmes negadījumu
ZP rezerves	Dati par reģistrētajām apdrošināšanas atlīdzību rezervēm
Lēmums	Dati par lēmumiem, kas pieņemti apr apdrošināšanas atlīdzības izmaksu
Maksājums	Dati par maksājumime, kas veikti par apdrošināšanas zaudējumu pieteikumu

2. pielikums. Datu noliktavas datu lauku apkopojums

Lauka nosaukums	Apraksts	Datu piemērs
AAS nosaukums	Apdrošināšanas sabiedrības nosaukums	"Cits apdrošinātājs"
BM subjekta adreses ATVK kods	Personas, kurai tika veikts BM aprēķins, ATVK klasifikātora kods	"010000"
BM subjekta dzimšanas datums	Personas, kurai tika veikts BM aprēķins, dzimšanas datums	"18.12.1921"
BM subjekta dzimums	Personas, kurai tika veikts BM aprēķins, dzimums	"V" "S" "n/z"
BM subjekta ISN	Personas, kurai tika veikts BM aprēķins, sistēmas iekšējs identifikators	"2 697 516"
BM subjekta personas veids	Personas, kurai tika veikts BM aprēķins, personas tips	"Fiziska persona" "Juridiska persona"
Līguma noslēgšanas kanāls	Vai līgums noslēgts klātienē vai attālināti	(n/z) Klātiene (Latvijā) Distances saziņa
Ir ārzemju polise	Polise derīga ārpus LV teritorijas	Jā Nē
Līguma veids (kods)	Apdrošināšanas līguma veida kods	'(n/z)' 'S' 'Z' 'R' 'A'
Polises ilgums (dienās)	Apdrošināšanas līguma ilgums dienās	185 365
Polises izdošanas datums	Apdrošināšanas līguma izdošanas datums	04.07.2008
Polises sākuma datums	Apdrošināšanas līguma sākuma datums	04.07.2008
ISN	Transportlīdzekļa identifikācijas nr. CSDD datu bāzē	12335654543
Izlaides gads	Transportlīdzekļa ražošanas gads	2015
Dzinēja tilpums cm2	Transportlīdzekļa dzinēja tilpums cm^2	1960
Degvielas tips	Transportlīdzekļa degvielas tips	'(n/z)' 'Dīzeļdegviela' 'Benzīns' 'Benzīns un gāze' 'Gāze'

Lauka nosaukums	Apraksts	Datu piemērs
		'Dīzeļdegviela un gāze' 'Elektrība'
Dzinēja jauda	Transportlīdzekļa dzinēja jauda kw	74 105
Pilna masa	Transportlīdzekļa pilna masa kg	
Krāsa	Transportlīdzekļa krāsa	'Rozā un vairākkrāsu' 'Gaiši zila un vairākkrāsu' 'Tumši pelēka un zila'
Marka	Transportlīdzekļa marka	SACNIA DAF
Modelis	Transportlīdzekļa modelis	318 X5
TRL kods	Transportlīdzekļa kods atbilstoši LTAB klasifikatoram	'V3I' 'V2I' 'V4I' 'K1I'
TRL vadītājs.Aktīvo brīdinājumu skaits	Transportl, ceļu policijā reģistrēto aktīvo brīdinājumu skaits	3
Dim - Transportlīdzeklis.Aktīvo soda punktu skaits	Personas, kurai tika veikts BM aprēķins, ceļu policijā reģistrēto soda punktu skaits	3
Vadītāja ISN	Vadītāja identifikātors	34454654564
Vadītāja adreses ATVK kods	Vadītāja adreses ATVK kods	010000
Vadītāja dzimums	Vadītāja dzimums	"V" "S" "n/z"
Vadītāja dzimšanas datums	Vadītāja dzimšanas datums	04.07.1992
CSNg apraksts	Ceļu satiksmes negadījuma apraksts. Apraksts var saturēt neadījuma vietas adresi, apstākļus un zaudējumu regulēšanas speciālistu komentārus	"SKRĪVERI, ŠOSEJA A6 77 KM."
CSNg datums	Ceļu satiksmes negadījuma datums	"09.11.2010"
CSNg iemesls - 1	Ceļu satiksmes negadījuma iemesls	'KRUST. PĀRBR. NOTEIK. NEIEVĒROŠANA' 'JOSLU MAIŅAS NOT. NEIEVĒROŠANA'

Lauka nosaukums	Apraksts	Datu piemērs
		'VADĪTĀJA NEUZMANĪBA' 'KUSTĪBA ATPAKAĻGAITĀ' 'NENOSKAIDROTS IEMESLS'
Lēmuma pieņemšanas datums	Lēmuma, par apdrošināšanas atlīdzības izmaksu pieņemšanas datums	01.06.2010
Pieteikuma datums	Zaudējuma pieteikuma datums	01.06.2010
Pieteikuma ID	Zaudējuma pieteikuma identifikātors	
Lēmuma veids	Lēmuma veids	'Izmaksa' 'Atteikums' 'Apņemšanās veikt izmaksu' 'Pakalpojums (serviss)'
CSNg novada ATVK kods	Ceļu satiksmes negadījuma ATVK kods	940000
CSNg skaits	Ceļu satiksmes negadījuma skaits	1
Rezerves summa (EUR)	Apdrošināšanas IBNR rezerves lielums	2 339,71
Lēmuma summa (EUR)	Apdrošināšanas atlīdzības summa EUR	1 262 318,12

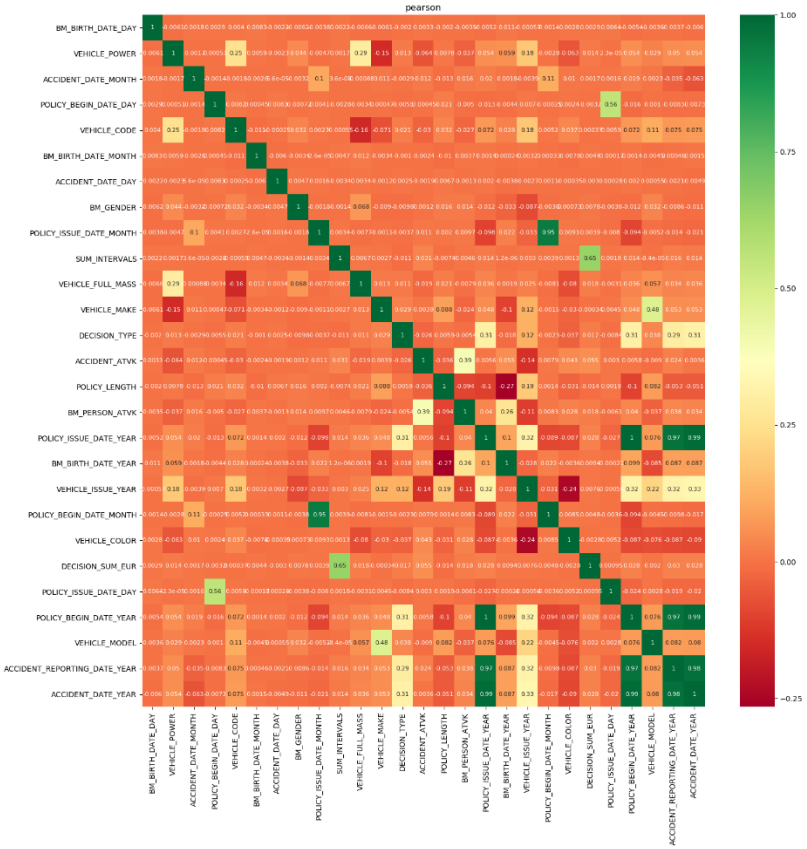
3. pielikums. Datu kolonu python tipi un netukšās vērtības

AAS kods	618737 non-null object
BM subjekta adreses ATVK kods	613916 non-null object
BM subjekta dzimšanas datums	411630 non-null object
BM subjekta dzimums	613916 non-null object
BM subjekta ISN	613916 non-null object
BM subjekta personas veids	613783 non-null object
CSNg apraksts	612702 non-null object
CSNg datums	618737 non-null object
CSNg iemesls - 1	3165 non-null object
CSNg iemesls - 2	158 non-null object
CSNg iemesls - 3	5 non-null object
Degvielas tips	615546 non-null object
Dim - Transportlīdzeklis.Aktīvo soda punktu skaits	77909 non-null float64
Dzinēja jauda	445995 non-null object
Dzinēja tilpums cm2	447585 non-null object
Ir ārzemju polise	618737 non-null object
Ir ilgtermiņa lieta	618737 non-null int64
ISN	582874 non-null object
Izlaides gads	585837 non-null object
Krāsa	568370 non-null object
Lēmuma pieņemšanas datums	618737 non-null object
Lēmuma veids	618737 non-null object
Līguma noslēgšanas kanāls	564354 non-null object
Līguma veids (kods)	618737 non-null object
Marka	610533 non-null object
Modelis	599862 non-null object
Pieteikuma datums	618737 non-null object
Pieteikuma ID	618737 non-null object
Pilna masa	578740 non-null object
Polises ilgums (dienās)	564354 non-null float64
Polises izdošanas datums	618737 non-null object
Polises sākuma datums	618737 non-null object
TRL BM subjekts.Aktīvo brīdinājumu skaits	613916 non-null float64

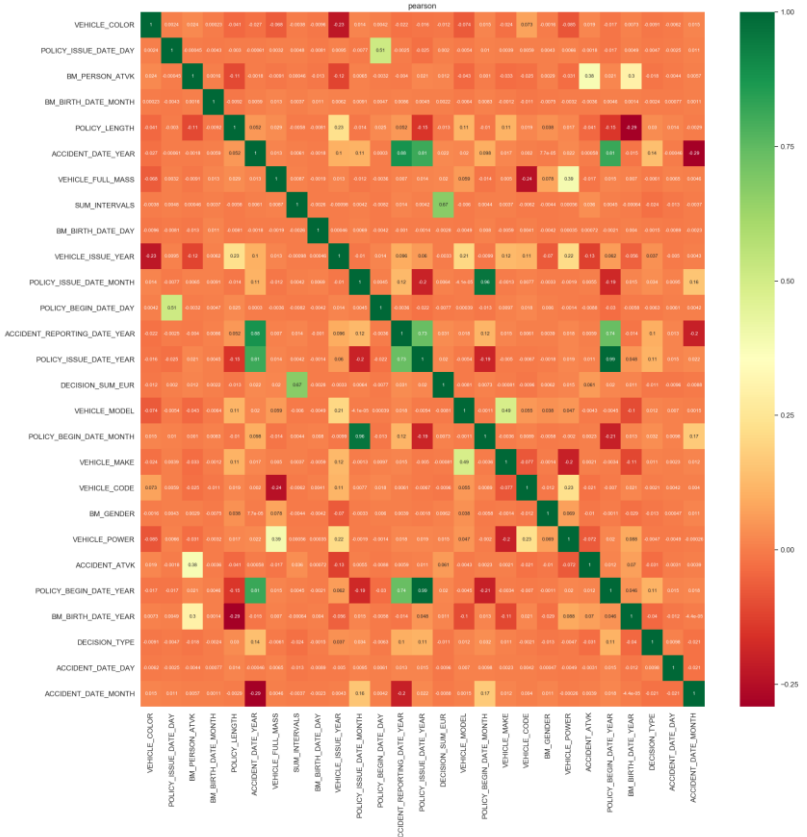
TRL BM subjekts.Aktīvo soda punktu skaits	613916 non-null float64
TRL kods	618737 non-null object
TRL vadītājs.Aktīvo brīdinājumu skaits	415669 non-null float64
Vadītāja adreses ATVK kods	415669 non-null object
Vadītāja dzimšanas datums	384564 non-null object
Vadītāja dzimums	415669 non-null object
Vadītāja ISN	415669 non-null object
CSNg novada ATVK kods	557058 non-null object
Valsts kods	618715 non-null object
Brīdinājumu skaits	49626 non-null float64
CSNg skaits	582837 non-null float64
Iepriekšējā rezerves summa (EUR)	600918 non-null object
Lēmuma summa (EUR)	546101 non-null object
Rezerves summa (EUR)	600918 non-null object
Soda punktu skaits	49626 non-null float64
Sūdzību skaits	2242 non-null float64

4. pielikums. Korelācijas koeficientu intensitātes kartes

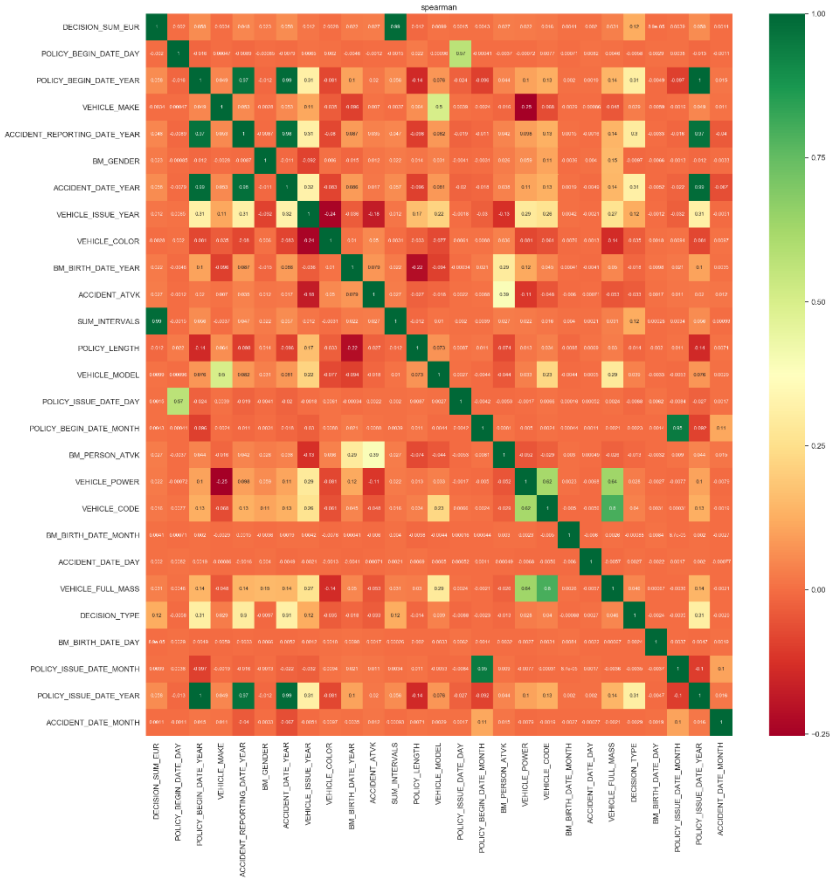
Pīrsona korelāciju matrica datiem sākot no 2008 gada



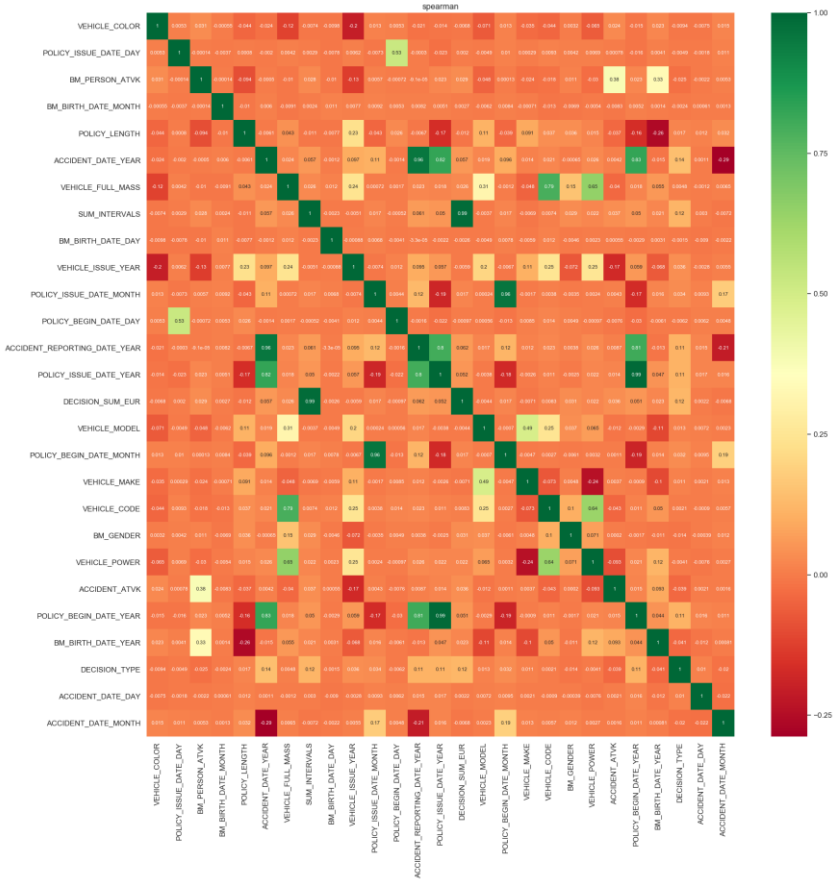
Pīrsona korelāciju matrica datiem sākot no 2016 gada



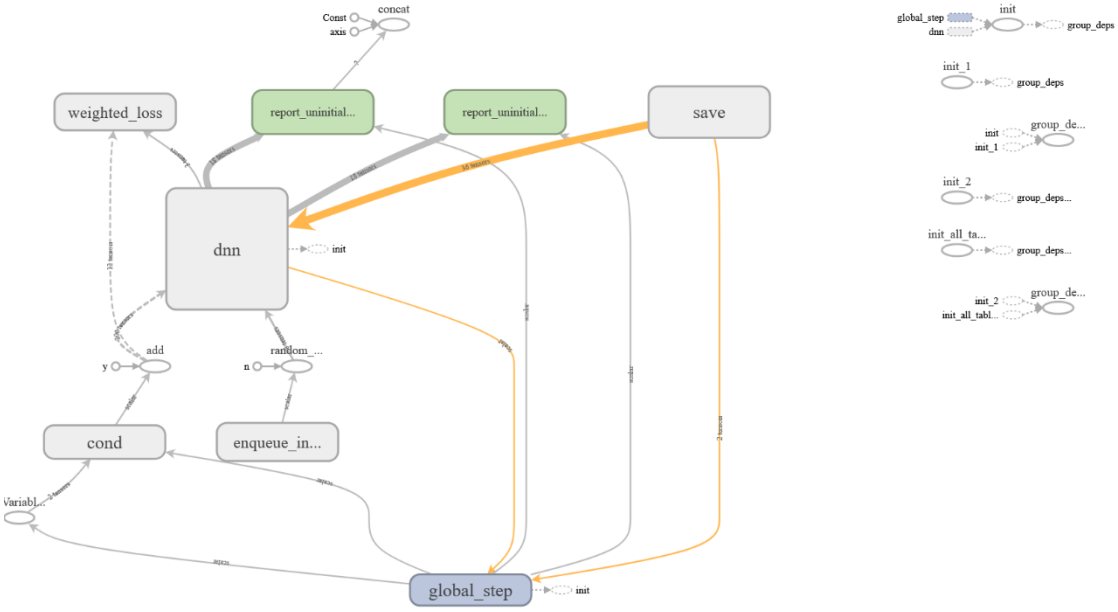
Spīrmana ranku korelāciju matrica datiem sākot no 2008 gada

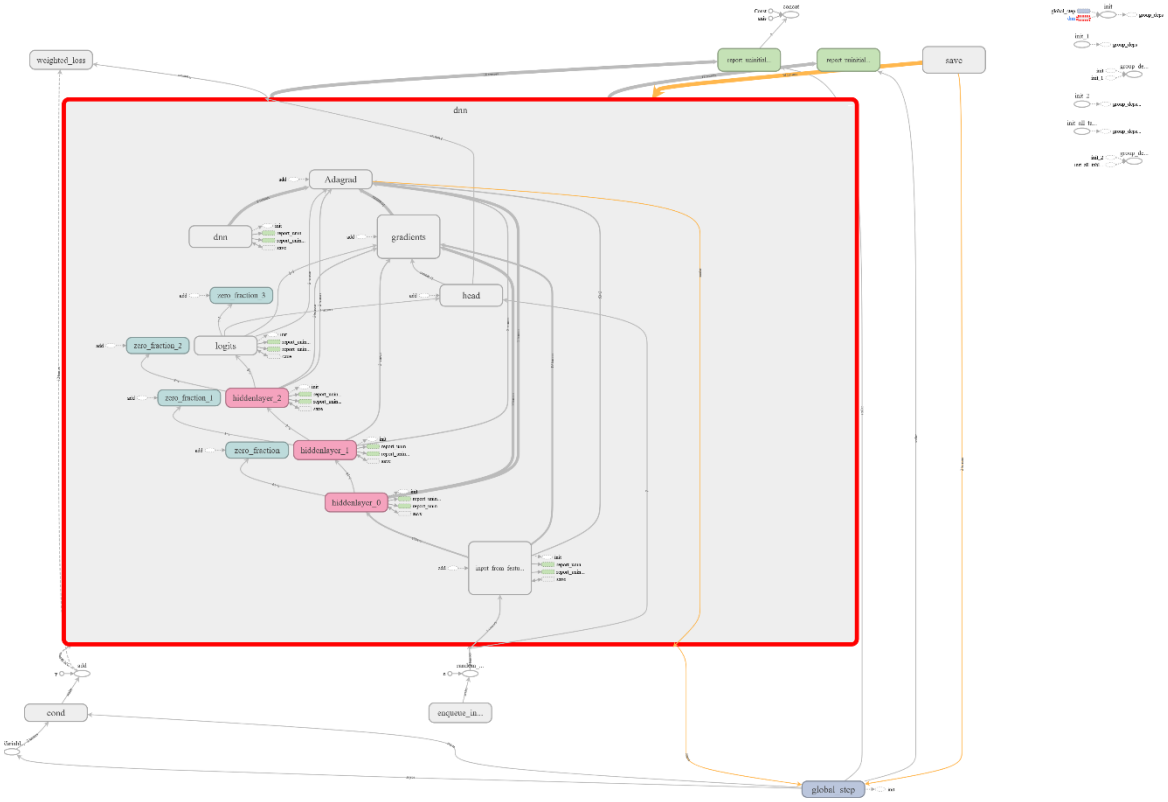


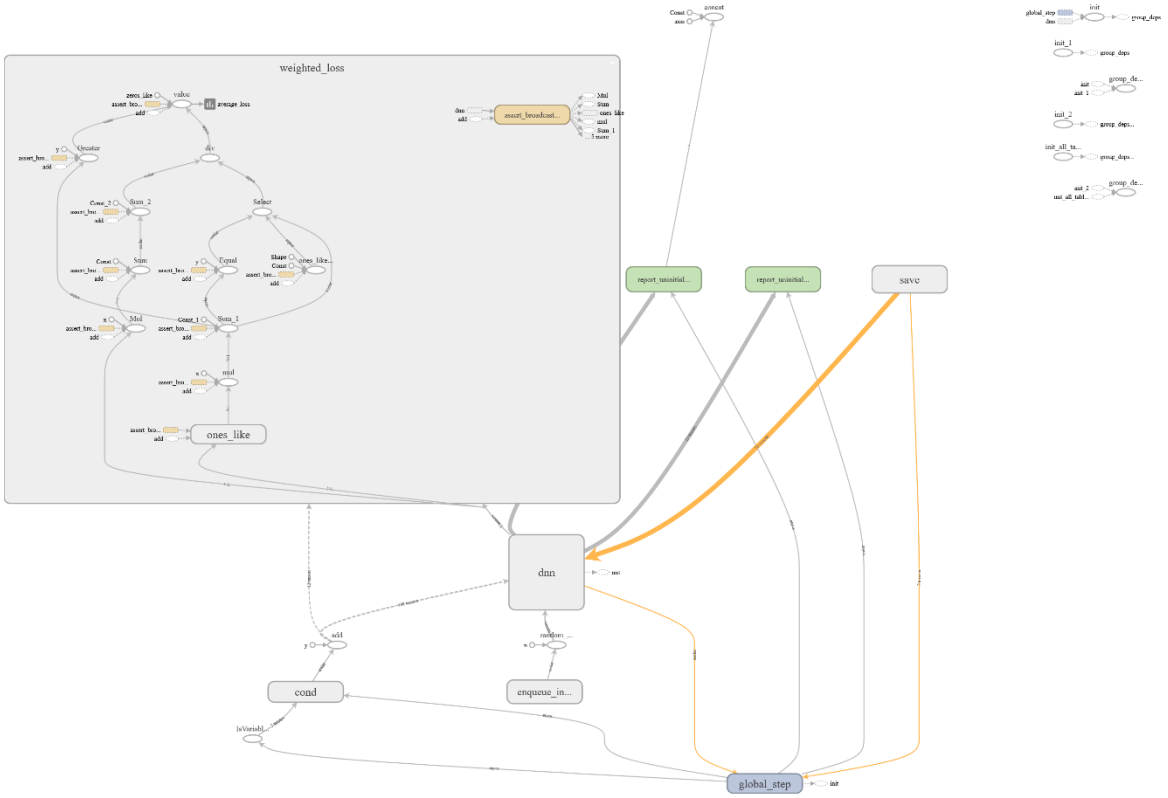
Spīrmana ranku korelāciju matrica datiem sākot no 2016 gada



5. pielikums. Izpildes grafi

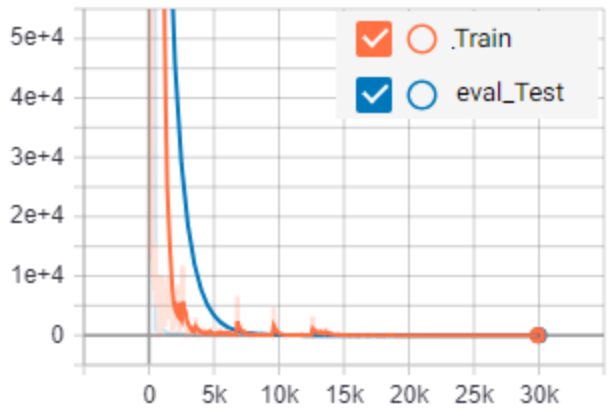


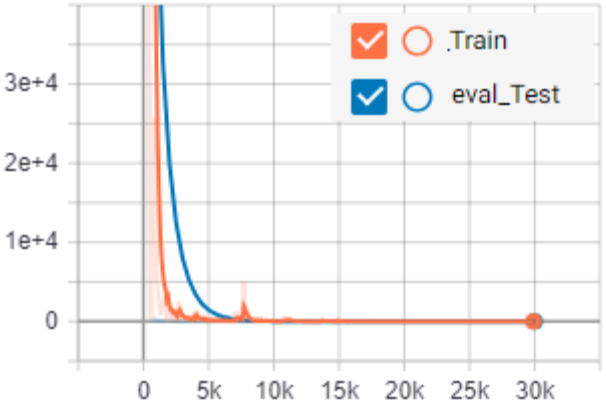



weighted_loss
assert_broadcast...
report_uninitial...
report_uninitial...
save
dnn
ones_like
cond
enqueue_in...
global_step

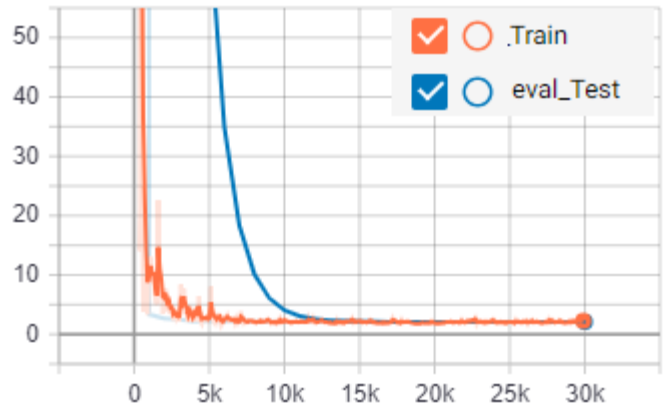
6. pielikums. Neironu tīkla modeļu zaudējumu funkcijas rezultāti

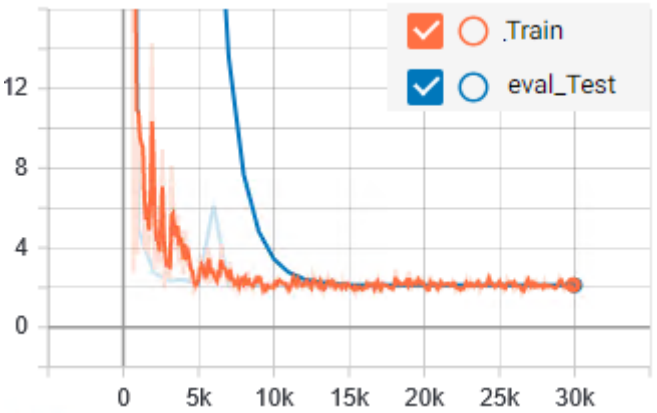
DNN regresija

Parametri	Vērtība	
optimizer	AdamOptimizer (learning_rate=0.001)	
activation_fn	tf.nn.lrelu	
hidden_units	[32, 12, 8, 16]	
num_epochs	10	
batch_size	8	
features	Visas	
Vidējā kļūda (treniņa dati):		**0.022**
Vidējā kļūda (testa dati):		**0.024**

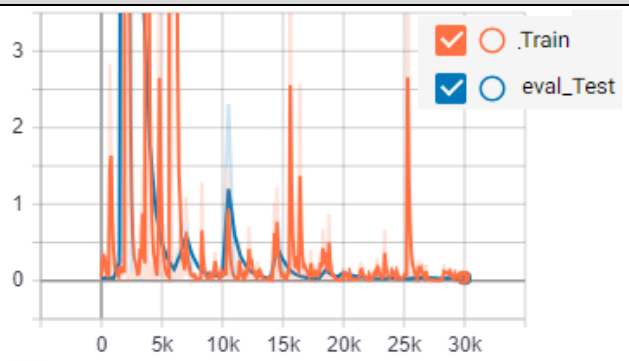
Parametri	Vērtība	
optimizer	AdamOptimizer (learning_rate=0.001)	
activation_fn	tf.nn.lrelu	
hidden_units	[32, 12, 8, 16]	
num_epochs	10	
batch_size	8	
features	Top 7	
Vidējā kļūda (treniņa dati):		**0.021**
Vidējā kļūda (testa dati):		**0.023**

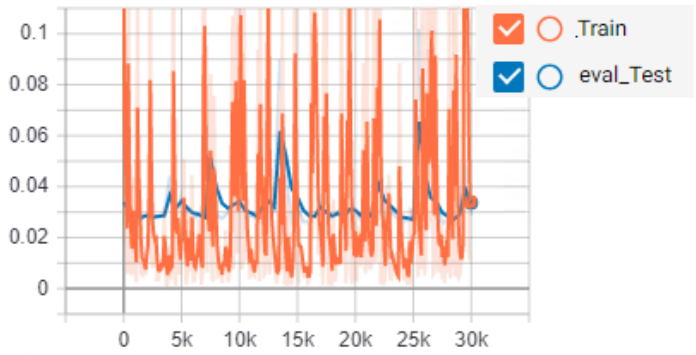
DNN klasifikācija

Parametri	Vērtība	
optimizer	AdamOptimizer (learning_rate=0.001)	
activation_fn	tf.nn.lrelu	
hidden_units	[32, 12, 8, 16]	
num_epochs	10	
batch_size	8	
features	Visas	
Vidējā kļūda (treniņa dati):		**2.09**
Vidējā kļūda (testa dati):		**2.11**

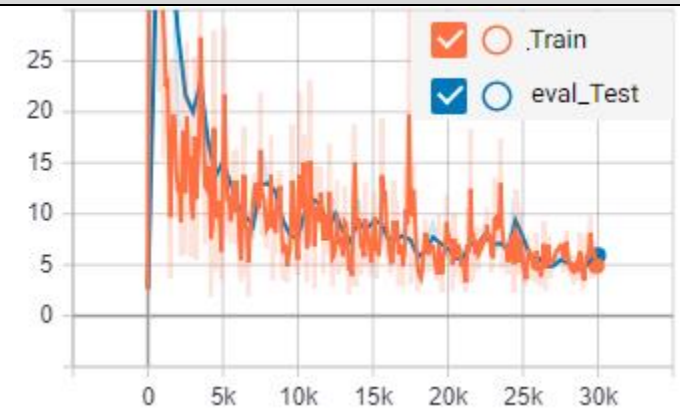
Parametri	Vērtība	
optimizer	AdamOptimizer (learning_rate=0.001)	
activation_fn	tf.nn.lrelu	
hidden_units	[32, 12, 8, 16]	
num_epochs	10	
batch_size	8	
features	Top 7	
Vidējā kļūda (treniņa dati):		**2.13**
Vidējā kļūda (testa dati):		**2.11**

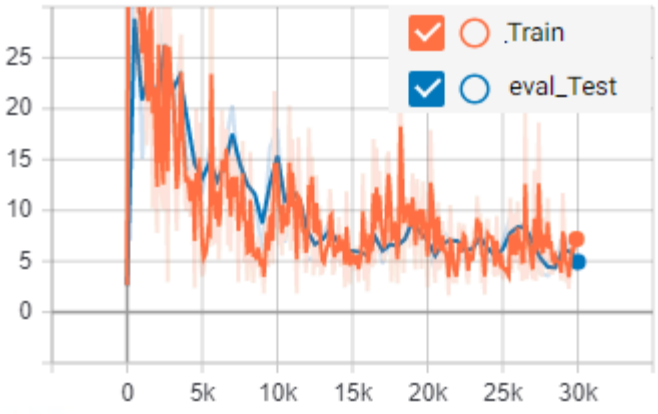
Lineāra regresija

Parametri	Vērtība	
optimizer	FtrlOptimizer (learning_rate=0.001)	
activation_fn		
hidden_units		
num_epochs	10	
batch_size	8	
features	Visas	
Vidējā kļūda (treniņa dati):		**0.0403**
Vidējā kļūda (testa dati):		**0.03988**

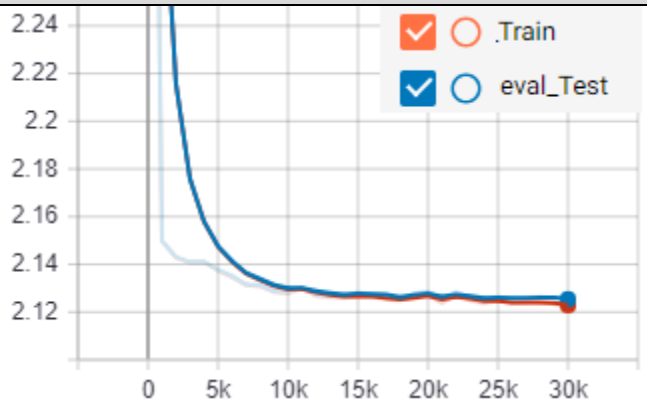
Parametri	Vērtība	
optimizer	FtrlOptimizer (learning_rate=0.001)	
activation_fn		
hidden_units		
num_epochs	10	
batch_size	8	
features	Top 7	
Vidējā kļūda (treniņa dati):		**0.02918**
Vidējā kļūda (testa dati):		**0.02862**

Lineāra klasifikācija

Parametri	Vērtība	
optimizer	FtrlOptimizer (learning_rate=0.001)	
activation_fn		
hidden_units		
num_epochs	10	
batch_size	8	
features	Visas	
Vidējā kļūda (treniņa dati):		**2.511**
Vidējā kļūda (testa dati):		**2.526**

Parametri	Vērtība	
optimizer	FtrlOptimizer (learning_rate=0.001)	
activation_fn		
hidden_units		
num_epochs	10	
batch_size	8	
features	Top 7	
Vidējā kļūda (treniņa dati):		**2.0104**
Vidējā kļūda (testa dati):		**2.0004**

RNN klasifikācija

Parametri	Vērtība	
optimizer	AdamOptimizer (learning_rate=0.001)	
hidden_units	[32, 12, 8, 16]	
num_epochs	10	
batch_size	8	
features	Visas	
Vidējā kļūda (treniņa dati):		**2.10**
Vidējā kļūda (testa dati):		**2.124**

7. pielikums. Modeļu treniņu un testu rezultātu piemēri

DNN regresija

Vehicle_make	Vehicle_model	Fact	Predicted_value
OPEL	ASTRA CARAVAN	1000	942.0041739940643
MAZDA	6	1000	942.0575201511383
SAAB	9-3	300	1154.7528207302094
AUDI	80 AVANT	400	1040.8033430576324
BMW	730	200	974.3083268404007
MERCEDES BENZ	A200	400	942.0181810855865
AUDI	A4 AVANT	300	998.571515083313
VOLVO	S80	400	942.0693665742874
RENAULT	MEGANE	400	1110.3277653455734
VW	PASSAT	400	1130.414456129074
TOYOTA	AVENSIS	200	942.0615434646606
VOLVO	S70	200	942.0393407344818
LAND ROVER	DISCOVERY	400	1130.3721368312836
AUDI	A4	1000	1006.8584233522415
OPEL	MERIVA	1000	1149.3314802646637
AUDI	100	1000	941.9744461774826
OPEL	ASTRA	500	941.9869631528854
AUDI	A4	300	1003.3150017261505
FORD	TRANSIT	1000	1111.6157472133636
OPEL	MERIVA	300	1111.4880442619324
MAZDA	626	500	1040.603592991829
MAZDA	626	300	941.9748187065125
AUDI	A3	1000	974.2172807455063
MITSUBISHI	GALANT	1000	942.0285373926163
VOLVO	S80	300	1114.4005507230759

DNN klasifikācija

Vehicle make	Vehicle model	Fact	Predicted value	Accuracy
MERCEDES BENZ	VANEO	2000	1000	0.0
OPEL	ASTRA CARAVAN	1000	1000	1.0
VW	PASSAT VARIANT	2000	1000	0.0
AUDI	A4 AVANT	1000	1000	1.0
OPEL	VECTRA CARAVAN	400	1000	0.0
BMW	520	200	1000	0.0
LEXUS	IS 250	300	1000	0.0
FORD	ESCORT	4000	1000	0.0
BMW	530	1000	1000	1.0
MERCEDES BENZ	CLK 270	1000	1000	1.0
AUDI	A4	200	1000	0.0

BMW	520	1000	1000	1.0
AUDI	80 AVANT	300	1000	0.0
OPEL	ASTRA	1000	1000	1.0
AUDI	RS6	500	1000	0.0
BMW	320	1000	1000	1.0
VW	PASSAT VARIANT	200	1000	0.0
AUDI	COUPE	1000	1000	1.0
VOLVO	S60	200	1000	0.0
OPEL	ASTRA	1000	1000	1.0
AUDI	A3	1000	1000	1.0
TOYOTA	LAND CRUISER	1000	1000	1.0
VW	SHARAN	200	1000	0.0
FORD	FOCUS	1000	1000	1.0
OPEL	VECTRA	1000	1000	1.0

Lineāra regresija

Vehicle_make	Vehicle_model	Fact	Predicted_value
VOLVO	V70	200	522.6939171552658
SAAB	9000	200	3605.274260044098
Å KODA	OCTAVIA	400	3922.0809936523438
TOYOTA	RAV4	1000	938.1448477506638
BMW	X5	1000	1064.7592693567276
FORD	S-MAX	1000	963.0353003740311
OPEL	ZAFIRA	1000	993.6515986919403
VW	PASSAT VARIANT	200	947.0710158348083
VW	TOURAN	4000	499.5045065879822
MERCEDES BENZ	310	200	953.7083655595779
AUDI	A4 AVANT	2000	605.1024794578552
PEUGEOT	307	400	562.5804886221886
TOYOTA	COROLLA	3000	1444.382518529892
MERCEDES BENZ	ML 270	500	994.633361697197
AUDI	A6 AVANT	400	2435.1145327091217
VW	PASSAT VARIANT	200	948.1484442949295
VW	SHARAN	2000	940.8960491418839
MERCEDES BENZ	ML 270	400	4220.423102378845
VW	PASSAT VARIANT	400	1124.9875277280807
RENAULT	LAGUNA	4000	3216.968774795532
OPEL	VECTRA CARAVAN	300	897.3030000925064
CITROEN	C3	200	927.1508455276489
VOLVO	V70	400	1113.2419109344482
OPEL	ZAFIRA	10000	3344.1954851150513
SEAT	TOLEDO	4000	1502.8975903987885

Lineāra klasifikācija

Vehicle_make	Vehicle_model	Fact	Predicted_value	Accuracy
PORSCHE	CAYENNE	500	1000	0.0
TOYOTA	COROLLA	100	200	0.0
VW	JETTA	10000	200	0.0
VW	GOLF VARIANT	100	200	0.0
BMW	530	2000	200	0.0
VOLVO	V70	200	200	1.0
VW	POLO	200	200	1.0
AUDI	A4	400	200	0.0
MAZDA	MPV	300	200	0.0
PEUGEOT	405	1000	200	0.0
PONTIAC	GRAND PRIX	500	400	0.0
BMW	318	2000	200	0.0
SAAB	9-5	300	200	0.0
FORD	FIESTA	500	200	0.0
VOLVO	V70	1000	200	0.0
FORD	MONDEO	300	200	0.0
OPEL	ASTRA	1000	200	0.0
VOLVO	V70	1000	400	0.0
HONDA	CIVIC	1000	200	0.0
LAND ROVER	DISCOVERY	100	200	0.0
VW	SHARAN	400	200	0.0
AUDI	A6	1000	1000	1.0
CITROEN	C3	400	200	0.0
MAZDA	626	400	200	0.0
VOLVO	XC70	300	200	0.0
HONDA	CIVIC	200	200	1.0
VW	PASSAT VARIANT	300	200	0.0
MERCEDES BENZ	SPRINTER 311	200	400	0.0
BMW	523	200	200	1.0
MAZDA	PREMACY	1000	200	0.0

8. pielikums. Zaudējumu noteikšanas klasifikācijas un regresijas rezultāti

Model type	Epochs	Steps max	Batch size	Optimizer	Layers	Activation	Dropout	Loss train (unnormalized)	Loss test (unnormalized)	Features count
DNNRegressor	2	5000	64	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.15	5849.085399	3338.945193	2
DNNRegressor	2	5000	64	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.15	4022.465165	7256.49447	2
DNNRegressor	2	5000	64	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.15	5227.855201	4528.007509	2
DNNRegressor	2	5000	64	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.15	5510.356516	4493.253737	2
DNNRegressor	2	5000	64	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.15	4067.834726	4402.827239	2
DNNRegressor	2	5000	64	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.15	4149.332586	4126.76756	2
DNNRegressor	10	500000	64	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.15	5596.82803	3069.408249	2
DNNRegressor	10	500000	64	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.15	5568.104345	3759.449694	2
DNNRegressor	10	500000	64	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.15	4491.746876	3472.428113	2
DNNRegressor	10	500000	64	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.15	3593.621293	5668.6377	2
DNNRegressor	10	500000	64	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.15	5626.698894	3971.9701	2
DNNRegressor	10	500000	64	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.15	3503.849501	7732.971927	2
DNNRegressor	10	500000	64	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.1	5836.828591	3145.157548	3
DNNRegressor	10	500000	64	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.1	4564.562191	6291.172228	3
DNNRegressor	10	500000	64	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.1	4750.060421	5889.038631	3
DNNRegressor	5	500000	64	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.1	5153.057927	4800.145414	4
DNNRegressor	5	500000	64	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.1	5333.95688	4466.200846	4
DNNRegressor	5	500000	64	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.1	4402.473623	2518.629985	4
DNNRegressor	5	500000	64	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.1	3967.144499	7167.51478	4
DNNRegressor	5	500000	64	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.1	5296.526782	4428.120115	4
DNNRegressor	5	500000	64	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.1	4081.048962	4369.230117	4
DNNRegressor	10	500000	64	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.1	3821.181996	7309.80561	6
DNNRegressor	10	500000	64	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.1	5277.544697	5132.223296	6
DNNRegressor	10	500000	64	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.1	3516.269608	7522.426898	6

DNNRegressor	10	500000	64	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.1	5129.302475	4553.223258	7
DNNRegressor	10	500000	64	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.1	5559.440028	4522.908076	7
DNNRegressor	10	500000	64	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.1	5875.303545	3508.558844	7
DNNRegressor	5	200000	4	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.1	5191.227447	4017.045211	19
DNNRegressor	5	200000	4	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.1	5107.021525	5657.782416	19
DNNRegressor	5	200000	4	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.1	5438.58765	4578.037925	19
DNNRegressor	5	200000	4	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.1	5432.805984	4405.314461	19
DNNRegressor	5	200000	4	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.1	5751.739389	3945.970603	7
DNNRegressor	5	200000	4	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.1	4420.090723	6811.979742	7
DNNRegressor	5	200000	4	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.1	5119.701749	5176.582077	7
DNNRegressor	5	200000	4	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.1	3993.894201	7422.674839	7
DNNRegressor	5	200000	4	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.1	5787.110323	2502.046731	7
DNNRegressor	5	200000	4	AdagradOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.1	5498.995657	4123.697769	7
DNNRegressor	5	200000	4	AdagradOptimizer(learning_rate=0.01)	[32, 16, 16, 8]	tf.nn.relu	0.1	5275.889921	4701.638833	19
DNNRegressor	5	200000	4	AdagradOptimizer(learning_rate=0.01)	[32, 16, 16, 8]	tf.nn.relu	0.1	4186.443324	4731.871721	19
DNNRegressor	5	200000	4	AdadeltaOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.1	4142.446117	7210.963141	19
DNNRegressor	5	200000	4	AdadeltaOptimizer(learning_rate=0.1)	[32, 16, 16, 8]	tf.nn.relu	0.1	5494.944915	4747.771292	19
DNNRegressor	5	200000	4	AdadeltaOptimizer(learning_rate=0.01)	[32, 16, 16, 8]	tf.nn.relu	0.1	6403.23824	6200.413214	19
DNNRegressor	6	300000	4	AdamOptimizer(learning_rate=0.01)	[128, 64, 16, 8]	tf.nn.relu	0.1	4310.387087	6678.699621	19
DNNRegressor	6	300000	4	AdamOptimizer(learning_rate=0.01)	[128, 64, 16, 8]	tf.nn.relu	0.1	4238.760609	6422.810201	19
DNNRegressor	6	500000	4	AdamOptimizer(learning_rate=0.01)	[128, 64, 16, 8]	tf.nn.relu	0.25	4605.672287	6229.660317	19
DNNRegressor	6	500000	4	AdamOptimizer(learning_rate=0.01)	[128, 64, 16, 8]	tf.nn.sigmoid	0.1	5398.865358	4929.229298	19
LinearReggressor	6	500000	4	tf.train.FtrlOptimizer(learning_rate=0.1)				4965.177361	5298.567622	19
LinearReggressor	5	10000	4	tf.train.FtrlOptimizer(learning_rate=0.1)				5782.461555	3985.353051	19
LinearReggressor	5	10000	4	tf.train.FtrlOptimizer(learning_rate=0.1)				5355.95317	4986.24519	19
LinearReggressor	5	30000	4	tf.train.FtrlOptimizer(learning_rate=0.1)				4288.921277	4896.062193	19
LinearReggressor	5	50000	4	tf.train.FtrlOptimizer(learning_rate=0.1)				4134.936654	7233.305389	19
LinearReggressor	5	30000	4	tf.train.AdamOptimizer(learning_rate=0.01)				15344.11485	114545.2914	19

9. pielikums. Pazīmju kolonu konfigurācija

Datu lauks	Tips un parametri
VEHICLE_MAKE	Tips: tf.feature_column.embedding_column hash_bucket_size = 1500
VEHICLE_MODEL	Tips: tf.feature_column.indicator_column hash_bucket_size = 7000
VEHICLE_COLOR	Tips: tf.feature_column.indicator_column hash_bucket_size = 322
VEHICLE_CODE	Tips: tf.feature_column.indicator_column hash_bucket_size = 60
VEHICLE_FULL_MASS	Tips: tf.feature_column.bucketized_column boundaries = No 0 līdz 50000 ar soli 500
VEHICLE_ISSUE_YEAR	Tips: tf.feature_column.bucketized_column boundaries = No 1900 līdz 2025 ar soli 5
VEHICLE_POWER	Tips: tf.feature_column.bucketized_column boundaries = No 0 līdz 5000 ar soli 80
POLICY_LENGTH	Tips: tf.feature_column.bucketized_column boundaries = No 1900 līdz 2025 ar soli 5
BM_GENDER	Tips: tf.feature_column.indicator_column vocabulary_list=["S", "V", "n/z"]
BM_PERSON_TYPE	Tips: tf.feature_column.indicator_column vocabulary_list=["(nav zināms)", "Fiziska persona", "Juridiska persona"]
BM_PERSON_ATVK	Tips: tf.feature_column.numeric_column
ACCIDENT_ATVK	Tips: tf.feature_column.numeric_column
BM_BIRTH_DATE_YEAR	Tips: tf.feature_column.numeric_column
BM_BIRTH_DATE_MONTH	Tips: tf.feature_column.numeric_column
BM_BIRTH_DATE_DAY	Tips: tf.feature_column.numeric_column
ACCIDENT_DATE_YEAR	Tips: tf.feature_column.numeric_column
ACCIDENT_DATE_MONTH	Tips: tf.feature_column.numeric_column
ACCIDENT_DATE_DAY	Tips: tf.feature_column.numeric_column
POLICY_ISSUE_DATE_YEAR	Tips: tf.feature_column.numeric_column
POLICY_BEGIN_DATE_MONTH	Tips: tf.feature_column.numeric_column

POLICY_BEGIN_DATE_DAY	Tips: tf.feature_column.numeric_column
ACCIDENT_REPORTING_DATE_YEAR	Tips: tf.feature_column.numeric_column
DECISION_TYPE	Tips: tf.feature_column.indicator_column vocabulary_list = ["Lietas administrēšanas izmaksas", "Regresa lietas administrēšanas izmaksas", "Apņemšanās veikt izmaksu", "Atteikums", "Daļēja izmaksa", "Izmaksa", "Lietas izbeigšana", "Lietas pārsūtīšana", "Lēmuma anulēšana", "Lēmuma maiņa", "Noilgums", "Norēķinu korekcija", "Pakalpojums (serviss)", "Periodiskā izmaksa (tikai personai)"]

10. pielikums. Rezervju aprēķina modeļu un konfigurācijas pirmkods

Main.py

```
import ModelFactory as f
import Configurations as c

"""
0 config -  FULL FEATURES LIST
1 config -  BIG NEURON count
2 config -  ONE FEATURE NEXT YEAR SUM PREDICTION
3 config -  DECISION TYPE PREDICTION
4 config -  The same as 1 with regular NEURON count
5 config -  For RNN network
"""
config = f.create_model(c.get_config()[5])

"""
1 - run DNN regressor
2 - run Linear regressor
3 - run DNN classifier
4 - run Baseline classifier
5 - run Linear classifier
6 - run DNN Linear Combined Regression
7 - run RNN Classifier
"""
f.run_model(model_id = 1, config = config, norm_factor = 1000)
#f.run_model(model_id = 2, config = config, norm_factor = 1000)
#f.run_model(model_id = 3, config = config, norm_factor = None)
#f.run_model(model_id = 4, config = config, norm_factor = None)
#f.run_model(model_id = 5, config = config, norm_factor = None)
#f.run_model(model_id = 6, config = config, norm_factor = 1000)
#f.run_model(model_id = 7, config = config, norm_factor = None)
```

Configurations.py

```
import tensorflow as tf

class Configuration:
    feature_column_names = []
    feature_columns = []
    train_dataset = None
    test_dataset = None
    predict_dataset = None

    hidden_units = []
    optimizer_dnn_regression = None
    optimizer_dnn_classifier = None
    optimizer_linear_regression = None
    optimizer_linear_classifier = None
    activation_fn = None
    dropout = None
    batch_size = None
    num_epochs = None
    steps = None

    label_vocabulary=[]
    label_column_name = None

    split_datasets_by_issue_year = False

    for_rnn = False

def get_config(index = 0):
    config_list = []

    ###############################################################################################
    # 0 config -  FULL FEATURES LIST
    c = Configuration()
    c.feature_column_names = ["VEHICLE_MAKE", "VEHICLE_MODEL", "VEHICLE_FULL_MASS", "VEHICLE_ISSUE_YEAR",
                              "VEHICLE_POWER", "VEHICLE_COLOR", "VEHICLE_CODE",
                              "BM_GENDER", "BM_PERSON_TYPE", "BM_PERSON_ATVK",
                              "BM_BIRTH_DATE_YEAR", "BM_BIRTH_DATE_MONTH", "BM_BIRTH_DATE_DAY",
                              "ACCIDENT_DATE_YEAR", "ACCIDENT_DATE_MONTH", "ACCIDENT_DATE_DAY",
                              "POLICY_ISSUE_DATE_YEAR", "POLICY_ISSUE_DATE_MONTH", "POLICY_ISSUE_DATE_DAY",
                              "POLICY_BEGIN_DATE_YEAR", "POLICY_BEGIN_DATE_MONTH", "POLICY_BEGIN_DATE_DAY",
                              "POLICY_LENGTH","ACCIDENT_REPORTING_DATE_YEAR","DECISION_TYPE","ACCIDENT_ATVK",
                              "RESERVE_SUM"
                              ]
    c.hidden_units = [128, 64, 16, 8]
    c.optimizer_dnn_regression = tf.train.AdamOptimizer(learning_rate=0.01)
    c.optimizer_linear_regression = tf.train.FtrlOptimizer(learning_rate=0.01)
    c.optimizer_dnn_classifier = c.optimizer_dnn_regression
    c.optimizer_baseline_classifier = c.optimizer_linear_regression
    c.optimizer_linear_classifier = c.optimizer_linear_regression
    c.activation_fn = tf.nn.relu
    c.dropout = 0.1
    c.batch_size = 16
    c.num_epochs = 5
    c.steps = 1000

    c.label_vocabulary = [0, 50, 100, 200, 300, 400, 500, 1000, 2000, 3000, 4000, 5000, 10000, 100000000]
    c.label_column_name = "SUM_INTERVALS"

    config_list.append(c)

    ###############################################################################################
    # 1 config -  BIG NEURON count
    c = Configuration()
```

```
c.feature_column_names = ["VEHICLE_MODEL", "BM_GENDER", "VEHICLE_FULL_MASS", "BM_BIRTH_DATE_YEAR"]
c.hidden_units = [4096, 128]
c.optimizer_dnn_regression = tf.train.AdamOptimizer(learning_rate=0.01)
c.optimizer_linear_regression = tf.train.FtrlOptimizer(learning_rate=0.01)
c.optimizer_dnn_classifier = c.optimizer_dnn_regression
c.optimizer_baseline_classifier = c.optimizer_linear_regression
c.optimizer_linear_classifier = c.optimizer_linear_regression
c.activation_fn = tf.nn.relu
c.dropout = 0.2
c.batch_size = 4
c.num_epochs = 5
c.steps = 6000

c.label_vocabulary = [0, 50, 100, 200, 300, 400, 500, 1000, 2000, 3000, 4000, 5000, 10000, 100000000]
c.label_column_name = "SUM_INTERVALS"

config_list.append(c)

#############################################################################################################
# 2 config -  ONE FEATURE NEXT YEAR PREDICTION
c = Configuration()
c.feature_column_names = ["VEHICLE_MODEL"]
c.hidden_units = [128, 64, 16, 8]
c.optimizer_dnn_regression = tf.train.AdamOptimizer(learning_rate=0.01)
c.optimizer_linear_regression = tf.train.FtrlOptimizer(learning_rate=0.01)
c.optimizer_dnn_classifier = c.optimizer_dnn_regression
c.optimizer_baseline_classifier = c.optimizer_linear_regression
c.optimizer_linear_classifier = c.optimizer_linear_regression
c.activation_fn = tf.nn.relu
c.dropout = 0.2
c.batch_size = 4
c.num_epochs = 5
c.steps = 30000

c.label_vocabulary = [0, 50, 100, 200, 300, 400, 500, 1000, 2000, 3000, 4000, 5000, 10000, 100000000]
c.label_column_name = "SUM_INTERVALS"

config_list.append(c)

#############################################################################################################
# 3 config -  DECISION TYPE PREDICTION
c = Configuration()
c.feature_column_names = ["VEHICLE_MODEL","DECISION_SUM_EUR"]
c.hidden_units = [128, 64, 16, 8]
c.optimizer_dnn_regression = tf.train.AdamOptimizer(learning_rate=0.01)
c.optimizer_linear_regression = tf.train.FtrlOptimizer(learning_rate=0.01)
c.optimizer_dnn_classifier = c.optimizer_dnn_regression
c.optimizer_baseline_classifier = c.optimizer_linear_regression
c.optimizer_linear_classifier = c.optimizer_linear_regression
c.activation_fn = tf.nn.relu
c.dropout = 0.2
c.batch_size = 4
c.num_epochs = 5
c.steps = 30000

c.label_vocabulary = ["Lietas administrēšanas izmaksas",
                      "Regresa lietas administrēšanas izmaksas",
                      "Apņemšanās veikt izmaksu",
                      "Atteikums",
                      "Daļēja izmaksa",
                      "Izmaksa",
                      "Lietas izbeigšana",
                      "Lietas pārsūtīšana",
                      "Lēmuma anulēšana",
                      "Lēmuma maiņa",
                      "Noilgums",
                      "Norēķinu korekcija",
                      "Pakalpojums (serviss)",
                      "Periodiskā izmaksa (tikai personai)"]
c.label_column_name = "DECISION_TYPE"

config_list.append(c)

#############################################################################################################
# 4 config -  The same as 1 with regular NEURON count
c = Configuration()
c.feature_column_names = ["VEHICLE_MODEL", "BM_GENDER", "VEHICLE_FULL_MASS", "BM_BIRTH_DATE_YEAR"]
c.hidden_units = [128, 64, 16]
c.optimizer_dnn_regression = tf.train.AdamOptimizer(learning_rate=0.01)
c.optimizer_linear_regression = tf.train.FtrlOptimizer(learning_rate=0.01)
c.optimizer_dnn_classifier = c.optimizer_dnn_regression
c.optimizer_baseline_classifier = c.optimizer_linear_regression
c.optimizer_linear_classifier = c.optimizer_linear_regression
c.activation_fn = tf.nn.relu
c.dropout = 0.2
c.batch_size = 4
c.num_epochs = 5
c.steps = 6000

c.label_vocabulary = [0, 50, 100, 200, 300, 400, 500, 1000, 2000, 3000, 4000, 5000, 10000, 100000000]
c.label_column_name = "SUM_INTERVALS"

config_list.append(c)

#############################################################################################################
# 5 config -  For RNN network
c = Configuration()
c.feature_column_names = ["VEHICLE_MAKE"]
c.hidden_units = [128, 64, 16]
c.hidden_units = [4096, 128, 64]
c.optimizer_dnn_regression = tf.train.AdamOptimizer(learning_rate=0.01)
c.optimizer_linear_regression = tf.train.FtrlOptimizer(learning_rate=0.01)
c.optimizer_dnn_classifier = c.optimizer_dnn_regression
c.optimizer_baseline_classifier = c.optimizer_linear_regression
c.optimizer_linear_classifier = c.optimizer_linear_regression
c.activation_fn = tf.nn.relu
```

```
    c.dropout = 0.1
    c.batch_size = 4
    c.num_epochs = 5
    c.steps = 10000

    c.label_vocabulary = [0, 50, 100, 200, 300, 400, 500, 1000, 2000, 3000, 4000, 5000, 10000, 100000000]
    c.label_column_name = "SUM_INTERVALS"

    c.split_datasets_by_issue_year = False
    c.for_rnn = True

    config_list.append(c)

    ###############################################################################################################
    # 6 config -  CUSTOM FEATURES LIST
    c = Configuration()

    c.feature_column_names = ["ACCIDENT_ATVK", "BM_PERSON_ATVK", "VEHICLE_FULL_MASS", "DECISION_TYPE"]
    c.hidden_units = [128, 64, 16, 8]
    c.optimizer_dnn_regression = tf.train.AdamOptimizer(learning_rate=0.01)
    c.optimizer_linear_regression = tf.train.FtrlOptimizer(learning_rate=0.01)
    c.optimizer_dnn_classifier = c.optimizer_dnn_regression
    c.optimizer_baseline_classifier = c.optimizer_linear_regression
    c.optimizer_linear_classifier = c.optimizer_linear_regression
    c.activation_fn = tf.nn.relu
    c.dropout = 0.2
    c.batch_size = 4
    c.num_epochs = 5
    c.steps = 30000

    c.label_vocabulary = [0, 50, 100, 200, 300, 400, 500, 1000, 2000, 3000, 4000, 5000, 10000, 100000000]
    c.label_column_name = "SUM_INTERVALS"

    c.split_datasets_by_issue_year = True
    c.for_rnn = False

    config_list.append(c)

    ###############################################################################################################
    # 7 config -  CUSTOM FEATURES LIST 2
    c = Configuration()
    #c.feature_column_names = ["ACCIDENT_ATVK", "BM_PERSON_ATVK", "VEHICLE_FULL_MASS","RESERVE_SUM", "VEHICLE_MODEL"]
    #c.feature_column_names = ["RESERVE_SUM", "VEHICLE_MODEL", "VEHICLE_FULL_MASS"]
    c.feature_column_names = ["VEHICLE_MAKE", "VEHICLE_MODEL", "VEHICLE_FULL_MASS", "VEHICLE_ISSUE_YEAR",
                              "VEHICLE_POWER", "VEHICLE_COLOR", "VEHICLE_CODE",
                              "BM_GENDER", "BM_PERSON_TYPE", "BM_PERSON_ATVK",
                              "BM_BIRTH_DATE_YEAR", "BM_BIRTH_DATE_MONTH", "BM_BIRTH_DATE_DAY",
                              "ACCIDENT_DATE_YEAR", "ACCIDENT_DATE_MONTH", "ACCIDENT_DATE_DAY",
                              "POLICY_ISSUE_DATE_YEAR", "POLICY_ISSUE_DATE_MONTH", "POLICY_ISSUE_DATE_DAY",
                              "POLICY_BEGIN_DATE_YEAR", "POLICY_BEGIN_DATE_MONTH", "POLICY_BEGIN_DATE_DAY",
                              "POLICY_LENGTH", "ACCIDENT_REPORTING_DATE_YEAR", "DECISION_TYPE", "ACCIDENT_ATVK",
                              "RESERVE_SUM"
                              ]
    c.hidden_units = [128, 64, 16]
    #c.hidden_units = [1024, 512, 256]
    #c.optimizer_dnn_regression = tf.train.AdamOptimizer(learning_rate=0.001)

    """
    c.optimizer_dnn_regression = tf.train.ProximalAdagradOptimizer(
        learning_rate=0.1,
        l1_regularization_strength=0.001
    )
    """

    c.optimizer_dnn_regression = lambda: tf.train.AdamOptimizer(
        learning_rate=tf.train.exponential_decay(
            learning_rate=0.1,
            global_step=tf.train.get_global_step(),
            decay_steps=10000,
            decay_rate=0.96))

    c.optimizer_linear_regression = tf.train.FtrlOptimizer(learning_rate=0.01)
    c.optimizer_dnn_classifier = c.optimizer_dnn_regression
    c.optimizer_baseline_classifier = c.optimizer_linear_regression
    c.optimizer_linear_classifier = c.optimizer_linear_regression
    c.activation_fn = tf.nn.relu
    c.dropout = 0.1
    c.batch_size = 8
    c.num_epochs = 10
    c.steps = 20000

    c.label_vocabulary = [0, 50, 100, 200, 300, 400, 500, 1000, 2000, 3000, 4000, 5000, 10000, 100000000]
    c.label_column_name = "SUM_INTERVALS"

    c.split_datasets_by_issue_year = False
    c.for_rnn = False

    config_list.append(c)

    ###############################################################################################################
    # 8 config -  CUSTOM FEATURES LIST 3
    c = Configuration()

    #c.feature_column_names = ["RESERVE_SUM", "VEHICLE_MODEL", "VEHICLE_FULL_MASS","ACCIDENT_ATVK", "BM_PERSON_ATVK",
"POLICY_LENGTH", "VEHICLE_POWER"]
    c.feature_column_names = ["VEHICLE_MAKE", "VEHICLE_MODEL", "VEHICLE_FULL_MASS", "VEHICLE_ISSUE_YEAR",
                              "VEHICLE_POWER", "VEHICLE_COLOR", "VEHICLE_CODE",
                              "BM_GENDER", "BM_PERSON_TYPE", "BM_PERSON_ATVK",
                              "BM_BIRTH_DATE_YEAR", "BM_BIRTH_DATE_MONTH", "BM_BIRTH_DATE_DAY",
                              "ACCIDENT_DATE_YEAR", "ACCIDENT_DATE_MONTH", "ACCIDENT_DATE_DAY",
                              "POLICY_ISSUE_DATE_YEAR", "POLICY_ISSUE_DATE_MONTH", "POLICY_ISSUE_DATE_DAY",
                              "POLICY_BEGIN_DATE_YEAR", "POLICY_BEGIN_DATE_MONTH", "POLICY_BEGIN_DATE_DAY",
                              "POLICY_LENGTH", "ACCIDENT_REPORTING_DATE_YEAR", "DECISION_TYPE", "ACCIDENT_ATVK", RESERVE_SUM"
                              ]

    c.hidden_units = [32, 12, 8, 16]
```

```
    c.optimizer_dnn_regression = tf.train.AdamOptimizer(learning_rate=0.001)

    c.optimizer_linear_regression = tf.train.FtrlOptimizer(learning_rate=0.001)
    c.optimizer_dnn_classifier = c.optimizer_dnn_regression
    c.optimizer_baseline_classifier = c.optimizer_linear_regression
    c.optimizer_linear_classifier = c.optimizer_linear_regression
    c.activation_fn = tf.nn.relu
    c.dropout = 0.1
    c.batch_size = 8
    c.num_epochs = 10
    c.steps = 20000

    c.label_vocabulary = [0, 50, 100, 200, 300, 400, 500, 1000, 2000, 3000, 4000, 5000, 10000, 100000000]
    c.label_column_name = "SUM_INTERVALS"

    c.split_datasets_by_issue_year = False
    c.for_rnn = False

    config_list.append(c)

    ########################################################################################################

    return config_list
```

ModelFactory.py

```
import os
import shutil as sh
import pandas as pd
import tensorflow as tf
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

import DNNRegressor as re
import LinearRegressor as lire
import DNNClassifier as cl
import BaselineClassifier as blcl
import LinearClassifier as licl
import DNNLinearCombinedRegressor as combre
import RNNClassifier as rnncl

import InsuranceDataset as ds

FILE_DIR = os.getcwd() + "\\..\\resources\data\\"
DATASET_FILE_PATH = FILE_DIR + "md_data.csv"

DELIMITER = ","

MODEL_DIR = "../model"

pd.set_option("display.max_rows", 100)
pd.set_option("display.max_columns", 100)
pd.set_option("display.width", None)
pd.set_option("max_colwidth", 150)

# Enable logging
tf.logging.set_verbosity(tf.logging.INFO)
#tf.logging.set_verbosity(tf.logging.DEBUG)


def create_model(config):
    """
    load pandas dataframe from csv
    """
    df = ds.load_dataset_from_file(data_file_path=DATASET_FILE_PATH, delimiter=DELIMITER)

    """
    feature columns
    """
    config.feature_columns = create_features_columns(feature_column_names=config.feature_column_names, for_rnn =
config.for_rnn)

    df = ds.clean_and_fix_data(data_frame=df, feature_column_names=config.feature_column_names,
                               label_column_names=[config.label_column_name], label_vocabulary=config.label_vocabulary)

    """
    Filter data
    """
    # keep last X yeasr only
    df = df[df["POLICY_ISSUE_DATE_YEAR"] >=2009]
    import scipy.stats as stats

    #drop out decision with zeros and outliers
    df = df[df["DECISION_SUM_EUR"] > 50]
    df = df[df["DECISION_SUM_EUR"] < 10000]
    #df = df[df["VEHICLE_MAKE"] =="BMW"]
    #df = df[df["DECISION_TYPE"].isin(["Izmaksa", "Pakalpojums (serviss)"])]
    df = df[df["DECISION_TYPE"].isin(["Izmaksa"])]

    #temp read
    #df = pd.read_csv(os.getcwd() + "\\..\\resources\data\\" + "sync.csv", sep=",")
    #df["SUM_INTERVALS"] = df["SUM_INTERVALS"].astype(int).astype(str)


    """
    print statistics
    """
    #print_field_statistics(data=df, field_name="DECISION_SUM_EUR")
    #print_field_statistics(data=df, field_name="SUM_INTERVALS")
    #print_field_statistics(data=df, field_name="DECISION_TYPE")
    #print_field_statistics(data=df, field_name="POLICY_LENGTH")
    #print_field_statistics(data=df, field_name="VEHICLE_POWER")
    #print_field_statistics(data=df, field_name="VEHICLE_FULL_MASS")
    #print_field_statistics(data=df, field_name="RESERVE_SUM")
```

```
    calculate_features_importance(data=df.copy(), feature_column_names = config.feature_column_names, label_column_name =
config.label_column_name)

    """
    Debug
    """
    print_nan_records(df)

    """
    get data sets for tensorflow
    """
    (config.train_dataset, config.test_dataset, config.predict_dataset) = ds.get_datasets_with_labels(df,
config.label_column_name,
                     predict_fraction = 0.1, train_fraction=0.7, seed=None, split_datasets_by_issue_year =
config.split_datasets_by_issue_year)
    return config

def run_model(model_id, config, norm_factor):
    # delete model dir, comment this if you would like restore from checkpoint
    sh.rmtree(MODEL_DIR, ignore_errors=True)

    predict_results = []

    # run DNNRegression model
    if model_id == 1 :
        predict_results = re.run_model(feature_columns = config.feature_columns, train_dataset = config.train_dataset,
                                       test_dataset=config.test_dataset, predict_dataset=config.predict_dataset,
model_dir=MODEL_DIR,
                                          hidden_units=config.hidden_units, optimizer=config.optimizer_dnn_regression,
                                          activation_fn=config.activation_fn, dropout=config.dropout,
                                          batch_size=config.batch_size, num_epochs=config.num_epochs, steps=config.steps,
                                          norm_factor=norm_factor)

    #Linear regression
    if model_id == 2 :
        predict_results = lire.run_model(feature_columns=config.feature_columns, train_dataset=config.train_dataset,
                                       test_dataset=config.test_dataset, predict_dataset=config.predict_dataset,
model_dir=MODEL_DIR,
                                            optimizer=config.optimizer_linear_regression,
                                          batch_size=config.batch_size, num_epochs=config.num_epochs, steps=config.steps,
                                          norm_factor=norm_factor)
    #DNN classifier
    if model_id == 3 :
        predict_results = cl.run_model(feature_columns=config.feature_columns, train_dataset=config.train_dataset,
                                       test_dataset=config.test_dataset, predict_dataset=config.predict_dataset,
model_dir=MODEL_DIR,
                                          hidden_units=config.hidden_units, optimizer=config.optimizer_dnn_classifier,
                                          activation_fn=config.activation_fn, dropout=config.dropout,
                                          batch_size=config.batch_size, num_epochs=config.num_epochs, steps=config.steps,
                                          label_vocabulary = config.label_vocabulary)
    #Baseline classifier
    if model_id == 4 :
        predict_results = blcl.run_model(train_dataset = config.train_dataset,
                                       test_dataset=config.test_dataset, predict_dataset=config.predict_dataset,
model_dir=MODEL_DIR,
                                          optimizer=config.optimizer_baseline_classifier,
                                          batch_size=config.batch_size, num_epochs=config.num_epochs, steps=config.steps,
                                       label_vocabulary = config.label_vocabulary)

    # Linear classifier
    if model_id == 5:
        predict_results = licl.run_model(feature_columns = config.feature_columns,
                                          train_dataset=config.train_dataset,
                                          test_dataset=config.test_dataset, predict_dataset=config.predict_dataset,
                                          model_dir=MODEL_DIR,
                                          optimizer=config.optimizer_linear_classifier,
                                          batch_size=config.batch_size, num_epochs=config.num_epochs,
                                          steps=config.steps,
                                          label_vocabulary=config.label_vocabulary)

    # run DNNLinearCombinedRegression model
    if model_id == 6 :
        predict_results = combre.run_model(feature_columns = config.feature_columns, train_dataset = config.train_dataset,
                                       test_dataset=config.test_dataset, predict_dataset=config.predict_dataset,
model_dir=MODEL_DIR,
                                          hidden_units=config.hidden_units, optimizer=config.optimizer_dnn_regression,
                                          activation_fn=config.activation_fn, dropout=config.dropout,
                                          batch_size=config.batch_size, num_epochs=config.num_epochs, steps=config.steps,
                                          norm_factor=norm_factor)
    #RNN classifier
    if model_id == 7 :
        predict_results = rnncl.run_model(feature_columns=config.feature_columns, train_dataset=config.train_dataset,
                                       test_dataset=config.test_dataset, predict_dataset=config.predict_dataset,
model_dir=MODEL_DIR,
                                          hidden_units=config.hidden_units, optimizer=config.optimizer_dnn_classifier,
                                          batch_size=config.batch_size, num_epochs=config.num_epochs, steps=config.steps,
                                          label_vocabulary = config.label_vocabulary)



    """
    Generate prediction output
    """
    arr_results = np.empty((0, 9))
    for i, prediction in enumerate(predict_results):
        vehicle_make = config.predict_dataset[0].iloc[i]["VEHICLE_MAKE"]
        vehicle_model = config.predict_dataset[0].iloc[i]["VEHICLE_MODEL"]

        reserve_sum = None
        if "RESERVE_SUM" in config.predict_dataset[0]:
            reserve_sum = config.predict_dataset[0].iloc[i]["RESERVE_SUM"]

        fact = config.predict_dataset[1].iloc[i]
        prediction_item = 0
        accuracy = 0
        probability = 0
```

```
        accuracy_error = 0
        predicted_value = None
        additional_info = None

        #Regression metrics
        if "predictions" in prediction:
            predicted_value = norm_factor * prediction["predictions"][0]
            accuracy = predicted_value/float(config.predict_dataset[1].iloc[i])

        #Classifiers metrics
        if "probabilities" in prediction:
            predicted_value = prediction["classes"][0].decode()
            class_id = prediction["class_ids"]
            accuracy = 1 if fact == predicted_value else 0
            probability = prediction["probabilities"][class_id][0]
            additional_info = "[" + ' '.join("{:.2f}".format(x) for x in prediction["probabilities"]) + "] " +
str(config.label_vocabulary)

        accuracy_error = abs(1 - accuracy)

        if i< 500000:
            arr_results = np.append(arr=arr_results,
                                values=[[vehicle_make, vehicle_model, reserve_sum, fact, predicted_value,
                                        accuracy, probability, accuracy_error, additional_info]],
                                axis = 0
                                )


    df_results_types= {
        "Vehicle_make":str,
        "Vehicle_model":str,
        "Reserve_sum":float,
        "Fact":str,
        "Predicted_value":str,
        "Accuracy":float,
        "Probability":float,
        "Accuracy_error":float,
        "Additional_info":str
    }
    df_results = pd.DataFrame(data=arr_results, columns=["Vehicle_make", "Vehicle_model", "Reserve_sum", "Fact",
                    "Predicted_value", "Accuracy", "Probability", "Accuracy_error", "Additional_info"])
    df_results = df_results.astype(df_results_types)

    print(df_results)
    print(df_results.describe())

    #export results to file
    df_results.to_csv(path_or_buf=FILE_DIR+"result_model_"+str(model_id)+".csv", sep=DELIMITER)


def create_features_columns(feature_column_names, for_rnn = False):
    ##########################
    # define feature columns
    ##########################
    feature_columns = []
    from tensorflow.contrib.feature_column import sequence_categorical_column_with_hash_bucket
    from tensorflow.contrib.feature_column import sequence_categorical_column_with_identity

    # VEHICLE_MAKE
    # embedding vector dimension should be the 4th root of the number of categories
    if "VEHICLE_MAKE" in feature_column_names:
        if for_rnn == True:
            feature_column_vehiclemake = tf.feature_column.embedding_column(
            sequence_categorical_column_with_hash_bucket(key="VEHICLE_MAKE", hash_bucket_size=1500),dimension=6)
        else:
            feature_column_vehiclemake = tf.feature_column.embedding_column(
            tf.feature_column.categorical_column_with_hash_bucket(key="VEHICLE_MAKE", hash_bucket_size=1500), dimension=6)
        feature_columns.append(feature_column_vehiclemake)

    # VEHICLE_MODEL
    if "VEHICLE_MODEL" in feature_column_names:
        if for_rnn == True:
            feature_column_vehiclemodel = tf.feature_column.embedding_column(
            sequence_categorical_column_with_hash_bucket(key="VEHICLE_MODEL", hash_bucket_size=7000),dimension=9)
        else:
            feature_column_vehiclemodel = tf.feature_column.indicator_column(
            tf.feature_column.categorical_column_with_hash_bucket(
                key="VEHICLE_MODEL", hash_bucket_size=7000)
        )
        feature_columns.append(feature_column_vehiclemodel)

    # VEHICLE_COLOR
    if "VEHICLE_COLOR" in feature_column_names:
        feature_column_vehiclecolor = tf.feature_column.indicator_column(
            tf.feature_column.categorical_column_with_hash_bucket(
                key="VEHICLE_COLOR", hash_bucket_size=322)
        )
        feature_columns.append(feature_column_vehiclecolor)

    # VEHICLE_CODE
    if "VEHICLE_CODE" in feature_column_names:
        feature_column_vehiclecode = tf.feature_column.indicator_column(
            tf.feature_column.categorical_column_with_hash_bucket(
                key="VEHICLE_CODE", hash_bucket_size=60)
        )
        feature_columns.append(feature_column_vehiclecode)

    # VEHICLE_FULL_MASS
    if "VEHICLE_FULL_MASS" in feature_column_names:
        if for_rnn == True:
            feature_column_vehiclefullmass = tf.feature_column.embedding_column(
                sequence_categorical_column_with_identity(
                    key="VEHICLE_FULL_MASS",
                    num_buckets=100000),dimension=10)
        else:
            feature_column_vehiclefullmass = tf.feature_column.bucketized_column(
```

```
        source_column=tf.feature_column.numeric_column(key="VEHICLE_FULL_MASS"),
        boundaries=np.arange(0, 50000, 500).tolist())
    feature_columns.append(feature_column_vehiclefullmass)

# VEHICLE_ISSUE_YEAR
if "VEHICLE_ISSUE_YEAR" in feature_column_names:
    feature_column_vehiclereleaseyear =  tf.feature_column.bucketized_column(
        source_column=tf.feature_column.numeric_column(key="VEHICLE_ISSUE_YEAR"),
        boundaries=np.arange(1900, 2025, 5).tolist())
    feature_columns.append(feature_column_vehiclereleaseyear)

# VEHICLE_POWER
if "VEHICLE_POWER" in feature_column_names:
    feature_column_vehiclepower =  tf.feature_column.bucketized_column(
        source_column=tf.feature_column.numeric_column(key="VEHICLE_POWER"),
        boundaries=np.arange(0, 5000, 80).tolist())
    feature_columns.append(feature_column_vehiclepower)

# POLICY_LENGTH
if "POLICY_LENGTH" in feature_column_names:
    feature_column_policylength =   tf.feature_column.bucketized_column(
        source_column=tf.feature_column.numeric_column(key="POLICY_LENGTH"),
        boundaries=np.arange(0, 400, 15).tolist())
    feature_columns.append(feature_column_policylength)

# BM_GENDER
if "BM_GENDER" in feature_column_names:
    feature_column_bmgender = tf.feature_column.indicator_column(
        tf.feature_column.categorical_column_with_vocabulary_list(
        key="BM_GENDER", vocabulary_list=["S", "V", "n/z"])
    )
    feature_columns.append(feature_column_bmgender)

# BM_PERSON_TYPE
if "BM_PERSON_TYPE" in feature_column_names:
    feature_column_bmpersontype = tf.feature_column.indicator_column(
        tf.feature_column.categorical_column_with_vocabulary_list(
            key="BM_PERSON_TYPE", vocabulary_list=["\(nav zināms\)", "Fiziska persona", "Juridiska persona"])
    )
    feature_columns.append(feature_column_bmpersontype)

# BM_PERSON_ATVK
if "BM_PERSON_ATVK" in feature_column_names:
    feature_column_bmpersonatvk= tf.feature_column.numeric_column(key="BM_PERSON_ATVK")
    feature_columns.append(feature_column_bmpersonatvk)

# ACCIDENT_ATVK
if "ACCIDENT_ATVK" in feature_column_names:
    feature_column_accidentatvk = tf.feature_column.numeric_column(key="ACCIDENT_ATVK")
    feature_columns.append(feature_column_accidentatvk)

# BM_BIRTH_DATE_YEAR
if "BM_BIRTH_DATE_YEAR" in feature_column_names:
    feature_column_birthdateyear =  tf.feature_column.numeric_column(key="BM_BIRTH_DATE_YEAR")
    feature_columns.append(feature_column_birthdateyear)

# BM_BIRTH_DATE_MONTH
if "BM_BIRTH_DATE_MONTH" in feature_column_names:
    feature_column_birthdatemonth =  tf.feature_column.numeric_column(key="BM_BIRTH_DATE_MONTH")
    feature_columns.append(feature_column_birthdatemonth)

# BM_BIRTH_DATE_DAY
if "BM_BIRTH_DATE_DAY" in feature_column_names:
    feature_column_birthdateday =  tf.feature_column.numeric_column(key="BM_BIRTH_DATE_DAY")
    feature_columns.append(feature_column_birthdateday)

# ACCIDENT_DATE_YEAR
if "ACCIDENT_DATE_YEAR" in feature_column_names:
    feature_column_accidentdateyear =  tf.feature_column.numeric_column(key="ACCIDENT_DATE_YEAR")
    feature_columns.append(feature_column_accidentdateyear)

# ACCIDENT_DATE_MONTH
if "ACCIDENT_DATE_MONTH" in feature_column_names:
    feature_column_accidentdatemonth =  tf.feature_column.numeric_column(key="ACCIDENT_DATE_MONTH")
    feature_columns.append(feature_column_accidentdatemonth)

# ACCIDENT_DATE_DAY
if "ACCIDENT_DATE_DAY" in feature_column_names:
    feature_column_accidentdateday =  tf.feature_column.numeric_column(key="ACCIDENT_DATE_DAY")
    feature_columns.append(feature_column_accidentdateday)

# POLICY_ISSUE_DATE_YEAR
if "POLICY_ISSUE_DATE_YEAR" in feature_column_names:
    feature_column_policyissuedateyear =  tf.feature_column.numeric_column(key="POLICY_ISSUE_DATE_YEAR")
    feature_columns.append(feature_column_policyissuedateyear)

# POLICY_ISSUE_DATE_MONTH
if "POLICY_ISSUE_DATE_MONTH" in feature_column_names:
    feature_column_policyissuedatemonth =  tf.feature_column.numeric_column(key="POLICY_ISSUE_DATE_MONTH")
    feature_columns.append(feature_column_policyissuedatemonth)

# POLICY_ISSUE_DATE_DAY
if "POLICY_ISSUE_DATE_DAY" in feature_column_names:
    feature_column_policyissuedateday =  tf.feature_column.numeric_column(key="POLICY_ISSUE_DATE_DAY")
    feature_columns.append(feature_column_policyissuedateday)

# POLICY_BEGIN_DATE_YEAR
if "POLICY_BEGIN_DATE_YEAR" in feature_column_names:
    feature_column_policybegindateyear = tf.feature_column.numeric_column(key="POLICY_BEGIN_DATE_YEAR")
    feature_columns.append(feature_column_policybegindateyear)

# POLICY_BEGIN_DATE_MONTH
if "POLICY_BEGIN_DATE_MONTH" in feature_column_names:
    feature_column_policybegindatemonth = tf.feature_column.numeric_column(key="POLICY_BEGIN_DATE_MONTH")
    feature_columns.append(feature_column_policybegindatemonth)
```

```
    # POLICY_BEGIN_DATE_DAY
    if "POLICY_BEGIN_DATE_DAY" in feature_column_names:
        feature_column_policybegindateday = tf.feature_column.numeric_column(key="POLICY_BEGIN_DATE_DAY")
        feature_columns.append(feature_column_policybegindateday)

    # ACCIDENT_REPORTING_DATE_YEAR
    if "ACCIDENT_REPORTING_DATE_YEAR" in feature_column_names:
        feature_column_policybegindateday = tf.feature_column.numeric_column(key="ACCIDENT_REPORTING_DATE_YEAR")
        feature_columns.append(feature_column_policybegindateday)

    # DECISION_TYPE
    if "DECISION_TYPE" in feature_column_names:
        feature_column_decisiontype = tf.feature_column.indicator_column(
            tf.feature_column.categorical_column_with_vocabulary_list(
                key="DECISION_TYPE",
                vocabulary_list=["Lietas administrēšanas izmaksas",
                                 "Regresa lietas administrēšanas izmaksas",
                                 "Apņemšanās veikt izmaksu",
                                 "Atteikums",
                                 "Daļēja izmaksa",
                                 "Izmaksa",
                                 "Lietas izbeigšana",
                                 "Lietas pārsūtīšana",
                                 "Lēmuma anulēšana",
                                 "Lēmuma maiņa",
                                 "Noilgums",
                                 "Norēķinu korekcija",
                                 "Pakalpojums (serviss)",
                                 "Periodiskā izmaksa (tikai personai)"])
        )
        feature_columns.append(feature_column_decisiontype)

    # DECISION_SUM_EUR
    if "DECISION_SUM_EUR" in feature_column_names:
        feature_column_decisionsumeur = tf.feature_column.numeric_column(key="DECISION_SUM_EUR")
        feature_columns.append(feature_column_decisionsumeur)

    # RESERVE_SUM
    if "RESERVE_SUM" in feature_column_names:
        if for_rnn == True:
            feature_column_reservesumeur = tf.feature_column.embedding_column(
                sequence_categorical_column_with_identity(
                    key="RESERVE_SUM",
                    num_buckets=10000000), dimension=5)
        else:
            feature_column_reservesumeur = tf.feature_column.embedding_column(
            tf.feature_column.bucketized_column(
            source_column=tf.feature_column.numeric_column(key="RESERVE_SUM"),
            boundaries=np.arange(0, 10000000, 100).tolist()), 13
            )
        feature_columns.append(feature_column_reservesumeur)

    return feature_columns

def print_nan_records(df):
    print("\n\n###### print_nan_records ######")
    print(df.info())
    # print data rows with nan values if exist
    if df.isnull().sum().sum() > 0:
        print()
        print("Records with nan values:")
        df_nan_values = df[df.isnull().any(axis=1)]
        print(df_nan_values)
        print()
    else:
        print("No records with nan values")
    print("###### print_nan_records END ######\n")

def print_field_statistics(data, field_name):
    print("\n\n######",field_name,"######")
    if field_name in data:
        print(data[field_name].describe())
        print(data.groupby([field_name]).count())
        print("Null values count:", data[field_name].isna().sum())


        data.hist(column=field_name, bins=50)
        plt.title("Histogramma")
        plt.show()

        data.hist(column=field_name, log=True, bins=50)
        plt.title("Histogramma log")
        plt.show()

        plt.boxplot(data[field_name])
        plt.show()
    else:
        print("Field ",field_name, " IS NOT INCLUDED in dataset. Specify it as feature or label column.")

    print("###### END ######\n")

def calculate_features_importance(data, feature_column_names, label_column_name):
    from sklearn.preprocessing import LabelEncoder

    if 'VEHICLE_MAKE' in data:
        data['VEHICLE_MAKE'] = LabelEncoder().fit_transform(data['VEHICLE_MAKE'])
    if 'VEHICLE_MODEL' in data:
        data['VEHICLE_MODEL'] = LabelEncoder().fit_transform(data['VEHICLE_MODEL'])
    if 'VEHICLE_COLOR' in data:
        data['VEHICLE_COLOR'] = LabelEncoder().fit_transform(data['VEHICLE_COLOR'])
    if 'VEHICLE_CODE' in data:
        data['VEHICLE_CODE'] = LabelEncoder().fit_transform(data['VEHICLE_CODE'])
    if 'BM_GENDER' in data:
        data['BM_GENDER'] = LabelEncoder().fit_transform(data['BM_GENDER'])
    if 'SUM_INTERVALS' in data:
        data['SUM_INTERVALS']=data["SUM_INTERVALS"].astype(float)
```

```
    if 'DECISION_TYPE' in data:
        data['DECISION_TYPE'] = LabelEncoder().fit_transform(data['DECISION_TYPE'])
    if 'BM_PERSON_TYPE' in data:
        data['BM_PERSON_TYPE'] = LabelEncoder().fit_transform(data['BM_PERSON_TYPE'].astype(str))


    X = data[feature_column_names]  # independent columns
    y = data[label_column_name]  # target column i.e price range
    y = y.astype(int)

    # get correlations heatmap of each features in dataset
    corrmat = data.corr()
    top_corr_features = corrmat.index

    sns.set(font_scale=1.1)
    # plot heat map
    plt.figure(figsize=(20, 20))
    g1 = sns.heatmap(data[top_corr_features].corr(method = 'pearson' ), annot=True, cmap="RdYlGn", annot_kws={"size": 8})
    plt.title('pearson')
    plt.show()

    plt.figure(figsize=(20, 20))
    g3  = sns.heatmap(data[top_corr_features].corr(method='spearman'), annot=True, cmap="RdYlGn", annot_kws={"size": 8})
    plt.title('spearman')
    plt.show()


    if len(data)>550000:
        X = data[feature_column_names+[label_column_name]].iloc[500000:550000,:]  # independent columns
        y = data[label_column_name].iloc[500000:550000,-1]  # target column i.e price range
        y = y.astype(int)

    from sklearn.feature_selection import SelectKBest
    from sklearn.feature_selection import chi2
    bestfeatures = SelectKBest(score_func=chi2, k='all')
    fit = bestfeatures.fit(X, y)
    dfscores = pd.DataFrame(fit.scores_)
    dfcolumns = pd.DataFrame(X.columns)
    # concat two dataframes for better visualization
    featureScores = pd.concat([dfcolumns, dfscores], axis=1)
    featureScores.columns = ['Specs', 'Score']  # naming the dataframe columns
    print(featureScores.nlargest(24, 'Score'))  # print 10 best features

    from sklearn.ensemble import ExtraTreesClassifier
    model = ExtraTreesClassifier()
    model.fit(X, y)
    print(model.feature_importances_)  # use inbuilt class feature_importances of tree based classifiers
    # plot graph of feature importances for better visualization
    feat_importances = pd.Series(model.feature_importances_, index=X.columns)
    feat_importances.nlargest(20).plot(kind='barh')
    plt.show()
```

InsuranceDataset.py

```
import pandas as pd
import os
import numpy as np

def load_dataset_from_file(data_file_path, delimiter):

    # check data file path
    if not os.path.exists(data_file_path):
        print("Data set file not exist: " + data_file_path)
        exit()
    else:
        print(os.getcwd())
        print("Data set file is found: " + data_file_path)

    # Load data into pandas dataframe
    df = pd.read_csv(data_file_path, sep=delimiter)

    column_names_mapping = \
        {
        "Lēmuma summa (EUR)":"DECISION_SUM_EUR",
        "Marka":"VEHICLE_MAKE",
        "Modelis":"VEHICLE_MODEL",
        "Izlaides gads":"VEHICLE_ISSUE_YEAR",
        "Pilna masa":"VEHICLE_FULL_MASS",
        "Dzinēja jauda":"VEHICLE_POWER",
        "Krāsa":"VEHICLE_COLOR",
        "TRL kods":"VEHICLE_CODE",
        "BM subjekta dzimšanas datums":"BM_BIRTH_DATE",
        "BM subjekta dzimums":"BM_GENDER",
        "BM subjekta personas veids":"BM_PERSON_TYPE",
        "BM subjekta adreses ATVK kods":"BM_PERSON_ATVK",
        "CSNg datums":"ACCIDENT_DATE",
        "CSNg novada ATVK kods":"ACCIDENT_ATVK",
        "Pieteikuma datums":"ACCIDENT_REPORTING_DATE",
        "Polises izdošanas datums":"POLICY_ISSUE_DATE",
        "Polises sākuma datums":"POLICY_BEGIN_DATE",
        "Polises ilgums (dienās)":"POLICY_LENGTH",
        "Lēmuma veids":"DECISION_TYPE",
        "Rezerves summa (EUR)": "RESERVE_SUM"
         }

    # rename columns
    df = df.rename(columns = column_names_mapping)

    return df

def clean_and_fix_data(data_frame, feature_column_names = [], label_column_names = [], label_vocabulary = []):
    #split dates into seperate fields
```

```
    data_frame["BM_BIRTH_DATE"] = pd.to_datetime(data_frame["BM_BIRTH_DATE"], format="%d.%m.%Y", errors='coerce')
    data_frame["BM_BIRTH_DATE_YEAR"] = data_frame["BM_BIRTH_DATE"].dt.year
    data_frame["BM_BIRTH_DATE_MONTH"] = data_frame["BM_BIRTH_DATE"].dt.month
    data_frame["BM_BIRTH_DATE_DAY"] = data_frame["BM_BIRTH_DATE"].dt.day

    data_frame["ACCIDENT_DATE"] = pd.to_datetime(data_frame["ACCIDENT_DATE"], format="%d.%m.%Y", errors='coerce')
    data_frame["ACCIDENT_DATE_YEAR"] = data_frame["ACCIDENT_DATE"].dt.year
    data_frame["ACCIDENT_DATE_MONTH"] = data_frame["ACCIDENT_DATE"].dt.month
    data_frame["ACCIDENT_DATE_DAY"] = data_frame["ACCIDENT_DATE"].dt.day

    data_frame["POLICY_ISSUE_DATE"] = pd.to_datetime(data_frame["POLICY_ISSUE_DATE"], format="%d.%m.%Y", errors='coerce')
    data_frame["POLICY_ISSUE_DATE_YEAR"] = data_frame["POLICY_ISSUE_DATE"].dt.year
    data_frame["POLICY_ISSUE_DATE_MONTH"] = data_frame["POLICY_ISSUE_DATE"].dt.month
    data_frame["POLICY_ISSUE_DATE_DAY"] = data_frame["POLICY_ISSUE_DATE"].dt.day

    data_frame["POLICY_BEGIN_DATE"] = pd.to_datetime(data_frame["POLICY_BEGIN_DATE"], format="%d.%m.%Y", errors='coerce')
    data_frame["POLICY_BEGIN_DATE_YEAR"] = data_frame["POLICY_BEGIN_DATE"].dt.year
    data_frame["POLICY_BEGIN_DATE_MONTH"] = data_frame["POLICY_BEGIN_DATE"].dt.month
    data_frame["POLICY_BEGIN_DATE_DAY"] = data_frame["POLICY_BEGIN_DATE"].dt.day

    data_frame["ACCIDENT_REPORTING_DATE"] = pd.to_datetime(data_frame["ACCIDENT_REPORTING_DATE"], format="%y.%d.%m",
errors='coerce')
    data_frame["ACCIDENT_REPORTING_DATE_YEAR"] = data_frame["ACCIDENT_REPORTING_DATE"].dt.year
    data_frame["ACCIDENT_REPORTING_DATE_MONTH"] = data_frame["ACCIDENT_REPORTING_DATE"].dt.month
    data_frame["ACCIDENT_REPORTING_DATE_DAY"] = data_frame["ACCIDENT_REPORTING_DATE"].dt.day

    """
    convert feature column data types
    """
    # convert VEHICLE_FULL_MASS
    data_frame["VEHICLE_FULL_MASS"] = data_frame["VEHICLE_FULL_MASS"].str.replace(' ', '')
    data_frame["VEHICLE_FULL_MASS"] = data_frame["VEHICLE_FULL_MASS"].str.replace('\xa0', '')
    data_frame["VEHICLE_FULL_MASS"] = data_frame["VEHICLE_FULL_MASS"].str.replace(',', '.')
    data_frame["VEHICLE_FULL_MASS"] = pd.to_numeric(data_frame["VEHICLE_FULL_MASS"])

    # convert VEHICLE_ISSUE_YEAR
    data_frame["VEHICLE_ISSUE_YEAR"] = data_frame["VEHICLE_ISSUE_YEAR"].str.replace(' ', '')
    data_frame["VEHICLE_ISSUE_YEAR"] = data_frame["VEHICLE_ISSUE_YEAR"].str.replace('\xa0', '')
    data_frame["VEHICLE_ISSUE_YEAR"] = data_frame["VEHICLE_ISSUE_YEAR"].str.replace(',', '.')
    data_frame["VEHICLE_ISSUE_YEAR"] = pd.to_numeric(data_frame["VEHICLE_ISSUE_YEAR"])

    # convert VEHICLE_POWER
    data_frame["VEHICLE_POWER"] = data_frame["VEHICLE_POWER"].str.replace(' ', '')
    data_frame["VEHICLE_POWER"] = data_frame["VEHICLE_POWER"].str.replace('\xa0', '')
    data_frame["VEHICLE_POWER"] = data_frame["VEHICLE_POWER"].str.replace(',', '.')
    data_frame["VEHICLE_POWER"] = pd.to_numeric(data_frame["VEHICLE_POWER"])

    #convert BM_PERSON_ATVK
    data_frame["BM_PERSON_ATVK"] = data_frame["BM_PERSON_ATVK"].replace('(n/z)', np.nan)
    data_frame["BM_PERSON_ATVK"] = pd.to_numeric(data_frame["BM_PERSON_ATVK"])

    #convert ACCIDENT_ATVK
    data_frame["ACCIDENT_ATVK"] = data_frame["ACCIDENT_ATVK"].replace('(n/z)', np.nan)
    data_frame["ACCIDENT_ATVK"] = pd.to_numeric(data_frame["ACCIDENT_ATVK"])

    # convert decison sums to number and remove nan
    # remove space
    data_frame["DECISION_SUM_EUR"] = data_frame["DECISION_SUM_EUR"].str.replace(' ', '')
    # remove unicode space
    data_frame["DECISION_SUM_EUR"] = data_frame["DECISION_SUM_EUR"].str.replace('\xa0', '')
    # change decimal separator to point
    data_frame["DECISION_SUM_EUR"] = data_frame["DECISION_SUM_EUR"].str.replace(',', '.')
    # convert to numeric
    data_frame["DECISION_SUM_EUR"] = pd.to_numeric(data_frame["DECISION_SUM_EUR"])

    # convert RESERVE_SUM
    data_frame["RESERVE_SUM"] = data_frame["RESERVE_SUM"].str.replace(' ', '')
    # remove unicode space
    data_frame["RESERVE_SUM"] = data_frame["RESERVE_SUM"].str.replace('\xa0', '')
    # change decimal separator to point
    data_frame["RESERVE_SUM"] = data_frame["RESERVE_SUM"].str.replace(',', '.')
    # convert to numeric
    data_frame["RESERVE_SUM"] = pd.to_numeric(data_frame["RESERVE_SUM"])

    # create discrete decision sum intevals for classification
    # !!! set max sum in label_vocabulary, because otherwise data will be cut to max sum)
    if(type(label_vocabulary[0]) == int):
        data_frame["SUM_INTERVALS"] = pd.IntervalIndex(pd.cut(data_frame["DECISION_SUM_EUR"], label_vocabulary)).right
    else:
        data_frame["SUM_INTERVALS"] = pd.IntervalIndex(pd.cut(data_frame["DECISION_SUM_EUR"],
[0,1000,10000,100000000])).right

    #fix DECISION_TYPE
    data_frame["DECISION_TYPE"] = data_frame["DECISION_TYPE"].str.replace('\\t', '')
    data_frame["DECISION_TYPE"] = data_frame["DECISION_TYPE"].str.strip()

    #remowe incorrect vehicle models
    data_frame = data_frame[data_frame["VEHICLE_MODEL"] != '#NZ']
    data_frame = data_frame[data_frame["VEHICLE_MODEL"] != '-']

    # remowe reserves less then 0
    data_frame = data_frame[data_frame["RESERVE_SUM"] >= 0]


    """
    Create final column set
    """
    # keep only needed columns
    # df = data_frame
    data_column_names=["VEHICLE_MAKE","VEHICLE_MODEL","SUM_INTERVALS","DECISION_SUM_EUR","POLICY_ISSUE_DATE_YEAR",
"DECISION_TYPE"]
    all_needed_column_names = list(set(feature_column_names + label_column_names + data_column_names))
    df = data_frame[all_needed_column_names]

    # drop rows with nan values
    df = df.dropna(subset = all_needed_column_names)
```

```
    """
    replace nan values
    """
    """
    df = df.fillna(value={'VEHICLE_MAKE':'', 'VEHICLE_MODEL':'', 'VEHICLE_ISSUE_YEAR': 0,
                          'VEHICLE_FULL_MASS':0,'VEHICLE_POWER':0,'BM_GENDER':'n/z','BM_PERSON_TYPE':0,
                          'BM_BIRTH_DATE_YEAR':0,'BM_BIRTH_DATE_MONTH':0,'BM_BIRTH_DATE_DAY':0})
    """


    """"
    adjust data types
    """
    if "VEHICLE_FULL_MASS" in all_needed_column_names:
        df["VEHICLE_FULL_MASS"] = df["VEHICLE_FULL_MASS"].astype(int)
    if "POLICY_LENGTH" in all_needed_column_names:
        df["POLICY_LENGTH"] = df["POLICY_LENGTH"].astype(int)
    if "VEHICLE_ISSUE_YEAR" in all_needed_column_names:
        df["VEHICLE_ISSUE_YEAR"] = df["VEHICLE_ISSUE_YEAR"].astype(int)
    if "VEHICLE_POWER" in all_needed_column_names:
        df["VEHICLE_POWER"] = df["VEHICLE_POWER"].astype(int)
    if "BM_BIRTH_DATE_YEAR" in all_needed_column_names:
        df["BM_BIRTH_DATE_YEAR"] = df["BM_BIRTH_DATE_YEAR"].astype(int)
    if "SUM_INTERVALS" in all_needed_column_names:
        df["SUM_INTERVALS"] = df["SUM_INTERVALS"].astype(int).astype(str)
    if "RESERVE_SUM" in all_needed_column_names:
        df["RESERVE_SUM"] = df["RESERVE_SUM"].astype(int)

    return df

def get_datasets_with_labels(data_frame, label_column_name, predict_fraction = 0.1, train_fraction = 0.7, seed = None,
split_datasets_by_issue_year = False):
    """
    Create and returns test,train and predict data sets

    Returns:
        A (train, test, predict) `Datasets`
    """

    #Shuffle the data and split to train/test/predict datasets
    np.random.seed(seed)

    if split_datasets_by_issue_year == True:
        predict_dataset = data_frame[data_frame["POLICY_ISSUE_DATE_YEAR"] >=2018]
    else:
        predict_dataset = data_frame.sample(frac=predict_fraction, random_state=seed)

    tmp_dataset = data_frame.drop(predict_dataset.index)
    train_dataset = tmp_dataset.sample(frac=train_fraction, random_state=seed)
    test_dataset = tmp_dataset.drop(train_dataset.index)

    # get label column
    train_label = train_dataset.pop(label_column_name)
    test_label = test_dataset.pop(label_column_name)
    predict_label = predict_dataset.pop(label_column_name)

    return (train_dataset,train_label),(test_dataset, test_label),(predict_dataset, predict_label)
```

DNNRegressor.py

```
import tensorflow_estimator as tf


#
# run tensoarboard from cmd: tensorboard --logdir=path_to_model_dir
#

def run_model(feature_columns, train_dataset, test_dataset, predict_dataset, model_dir ,
              hidden_units=[], optimizer=None,
              activation_fn=None, dropout=None, batch_size=None, num_epochs=None, steps=None,
              norm_factor = None):
    """
    create model
    """

    model = tf.estimator.DNNRegressor(
        feature_columns=feature_columns,
        hidden_units=hidden_units,
        optimizer=optimizer,
        activation_fn=activation_fn,
        dropout=dropout,
        model_dir=model_dir,  # none - used default

        config = tf.estimator.RunConfig(
            save_checkpoints_steps=500,  # Save checkpoints every x steps.
            keep_checkpoint_max=10000  # Retain the x most recent checkpoints.
        )
    )

    print("Train dataset length:",len(train_dataset[0]))


    #train model
    train_result = model.train(
        input_fn = tf.estimator.inputs.pandas_input_fn(
            x = train_dataset[0],
            y = train_dataset[1].astype(float) / norm_factor,
            shuffle=True,
            batch_size=batch_size,
            num_epochs = num_epochs), steps = steps)
```

```
    from tensorflow.python.training import checkpoint_management
    for ckpt_file in checkpoint_management.get_checkpoint_state(checkpoint_dir = model_dir).all_model_checkpoint_paths:
        eval_result_test = model.evaluate(input_fn=tf.estimator.inputs.pandas_input_fn(
            x=test_dataset[0],
            y=test_dataset[1].astype(float) / norm_factor,
            shuffle=False),
            name="Test", checkpoint_path=ckpt_file)

        eval_result_train = model.evaluate(input_fn=tf.estimator.inputs.pandas_input_fn(
            x=train_dataset[0],
            y=train_dataset[1].astype(float) / norm_factor,
            shuffle=False), name="Train", checkpoint_path=ckpt_file)

    # Evaluate how the model performs on test dataset for last checkpoint.
    eval_result_train = model.evaluate(input_fn = tf.estimator.inputs.pandas_input_fn(
        x = train_dataset[0],
        y = train_dataset[1].astype(float) / norm_factor ,
        shuffle=False),
        name = "Train Final")

    eval_result_test = model.evaluate(input_fn = tf.estimator.inputs.pandas_input_fn(
        x = test_dataset[0],
        y = test_dataset[1].astype(float) / norm_factor,
        shuffle=False),
        name = "Test Final")

    # print loss. Average loss, Average quadratic loss, Loss are the same characteristic.
    average_loss_train = eval_result_train["average_loss"]
    average_loss_test = eval_result_test["average_loss"]
    print("Average_loss raw: Train:{0} Test:{1}".format(average_loss_train, average_loss_test))
    print("Average_loss: Train:{0} Test:{1}".format(norm_factor * average_loss_train ** 0.5, norm_factor * average_loss_test
** 0.5))

    #prediction
    predict_results = model.predict(
        input_fn=tf.estimator.inputs.pandas_input_fn(x = predict_dataset[0], shuffle=False),
        predict_keys=None
    )

    return predict_results
```

LinearRegressor.py

```
import tensorflow_estimator as tf

def run_model(feature_columns, train_dataset, test_dataset, predict_dataset, model_dir ,
              optimizer=None, batch_size=None, num_epochs=None, steps=None,  norm_factor = None):
    """
    create model
    """
    #simple model

    model = tf.estimator.LinearRegressor(
        feature_columns = feature_columns,
        optimizer=optimizer,
        model_dir= model_dir, # none - used default

        config = tf.estimator.RunConfig(
            save_checkpoints_steps=500,  # Save checkpoints every x steps.
            keep_checkpoint_max=10000  # Retain the x most recent checkpoints.
        )
    )

    print("Train dataset length:",len(train_dataset[0]))

    #train model
    model.train(
        input_fn = tf.estimator.inputs.pandas_input_fn(
            x = train_dataset[0],
            y = train_dataset[1].astype(float) / norm_factor,
            shuffle=True,
            batch_size=batch_size,
            num_epochs = num_epochs), steps = steps)

    from tensorflow.python.training import checkpoint_management
    for ckpt_file in checkpoint_management.get_checkpoint_state(checkpoint_dir=model_dir).all_model_checkpoint_paths:
        eval_result_test = model.evaluate(input_fn=tf.estimator.inputs.pandas_input_fn(
            x=test_dataset[0],
            y=test_dataset[1].astype(float) / norm_factor,
            shuffle=False),
            name="Test", checkpoint_path=ckpt_file)

        eval_result_train = model.evaluate(input_fn=tf.estimator.inputs.pandas_input_fn(
            x=train_dataset[0],
            y=train_dataset[1].astype(float) / norm_factor,
            shuffle=False), name="Train", checkpoint_path=ckpt_file)

    # Evaluate how the model performs on test dataset for last checkpoint.
    eval_result_train = model.evaluate(input_fn=tf.estimator.inputs.pandas_input_fn(
        x=train_dataset[0],
        y=train_dataset[1].astype(float) / norm_factor,
        shuffle=False),
        name="Train Final")

    eval_result_test = model.evaluate(input_fn=tf.estimator.inputs.pandas_input_fn(
        x=test_dataset[0],
        y=test_dataset[1].astype(float) / norm_factor,
        shuffle=False),
        name="Test Final")

    # print loss. Average loss, Average quadratic loss, Loss are the same characteristic.
    average_loss_train = eval_result_train["average_loss"]
    average_loss_test = eval_result_test["average_loss"]
```

```
    print("Average_loss: Train:{0} Test:{1}".format(norm_factor * average_loss_train**0.5, norm_factor *
average_loss_test**0.5))


    #prediction
    predict_results = model.predict(
        input_fn=tf.estimator.inputs.pandas_input_fn(x = predict_dataset[0], shuffle=False),
        predict_keys=None
    )

    return predict_results
```

DNNClassifier.py

```
import tensorflow_estimator as tf

def run_model(feature_columns, train_dataset, test_dataset, predict_dataset, model_dir ,
            hidden_units=None, optimizer=None,
            activation_fn=None, dropout = None, batch_size=None, num_epochs = None, steps = None,
            label_vocabulary = []):
    """
    create model
    """

    label_vocabulary = [str(x) for x in label_vocabulary]

    model = tf.estimator.DNNClassifier(
        feature_columns = feature_columns,
        hidden_units = hidden_units,
        optimizer = optimizer,
        label_vocabulary = label_vocabulary,
        n_classes = len(label_vocabulary),
        activation_fn = activation_fn,
        dropout =  dropout,
        model_dir= model_dir,  # none - used default

        config = tf.estimator.RunConfig(
            save_checkpoints_steps=1000,  # Save checkpoints every x steps.
            keep_checkpoint_max=10000  # Retain the x most recent checkpoints.
        )
    )

    print("Train dataset length:",len(train_dataset[0]))

    #train model
    model.train(
        input_fn = tf.estimator.inputs.pandas_input_fn(
            x = train_dataset[0],
            y = train_dataset[1],
            shuffle = True,
            batch_size = batch_size,
            num_epochs = num_epochs), steps = steps)

    from tensorflow.python.training import checkpoint_management
    for ckpt_file in checkpoint_management.get_checkpoint_state(checkpoint_dir = model_dir).all_model_checkpoint_paths:
        eval_result_test = model.evaluate(input_fn=tf.estimator.inputs.pandas_input_fn(
            x=test_dataset[0],
            y=test_dataset[1],
            shuffle=False),
            name="Test", checkpoint_path=ckpt_file)

        eval_result_train = model.evaluate(input_fn=tf.estimator.inputs.pandas_input_fn(
            x=train_dataset[0],
            y=train_dataset[1],
            shuffle=False), name="Train", checkpoint_path=ckpt_file)

    # Evaluate how the model performs on test dataset for last checkpoint.
    eval_result_train = model.evaluate(input_fn = tf.estimator.inputs.pandas_input_fn(
        x = train_dataset[0],
        y = train_dataset[1],
        shuffle=False),
        name = "Train Final")

    eval_result_test = model.evaluate(input_fn = tf.estimator.inputs.pandas_input_fn(
        x = test_dataset[0],
        y = test_dataset[1],
        shuffle=False),
        name = "Test Final")

    # print loss. Average loss, Average quadratic loss, Loss are the same characteristic.
    average_loss_train = eval_result_train["average_loss"]
    average_loss_test = eval_result_test["average_loss"]
    print("Average_loss: Train:{0} Test:{1}".format(average_loss_train**0.5,  average_loss_test**0.5))

    #prediction
    predict_results = model.predict(
        input_fn=tf.estimator.inputs.pandas_input_fn(x = predict_dataset[0], shuffle=False),
        predict_keys= None
    )

    return predict_results
```

RNNClassifier.py

```
import tensorflow_estimator as tf

def run_model(feature_columns, train_dataset, test_dataset, predict_dataset, model_dir ,
            hidden_units=None, optimizer=None, batch_size=None, num_epochs = None, steps = None,
            label_vocabulary = []):
    """
```

```
    create model
    """

    label_vocabulary = [str(x) for x in label_vocabulary]

    from tensorflow_estimator.contrib.estimator import RNNClassifier
    model = RNNClassifier(
        sequence_feature_columns = feature_columns,
        num_units = hidden_units,
        cell_type='lstm',
        optimizer=optimizer,
        label_vocabulary=label_vocabulary,
        n_classes=len(label_vocabulary),
        model_dir=model_dir,  # none - used default

        config = tf.estimator.RunConfig(
            save_checkpoints_steps=1000,  # Save checkpoints every x steps.
            keep_checkpoint_max=10000  # Retain the x most recent checkpoints.
        )  # none - used default
    )

    print("Train dataset length:",len(train_dataset[0]))

    #train model
    model.train(
        input_fn = tf.estimator.inputs.pandas_input_fn(
            x = train_dataset[0],
            y = train_dataset[1],
            shuffle = True,
            batch_size = batch_size,
            num_epochs = num_epochs), steps = steps)

    from tensorflow.python.training import checkpoint_management
    for ckpt_file in checkpoint_management.get_checkpoint_state(checkpoint_dir=model_dir).all_model_checkpoint_paths:
        eval_result_test = model.evaluate(input_fn=tf.estimator.inputs.pandas_input_fn(
            x=test_dataset[0],
            y=test_dataset[1],
            shuffle=False),
            name="Test", checkpoint_path=ckpt_file)

        eval_result_train = model.evaluate(input_fn=tf.estimator.inputs.pandas_input_fn(
            x=train_dataset[0],
            y=train_dataset[1],
            shuffle=False), name="Train", checkpoint_path=ckpt_file)

    # Evaluate how the model performs on test dataset for last checkpoint.
    eval_result_train = model.evaluate(input_fn=tf.estimator.inputs.pandas_input_fn(
        x=train_dataset[0],
        y=train_dataset[1],
        shuffle=False),
        name="Train Final")

    eval_result_test = model.evaluate(input_fn=tf.estimator.inputs.pandas_input_fn(
        x=test_dataset[0],
        y=test_dataset[1],
        shuffle=False),
        name="Test Final")

    # print loss. Average loss, Average quadratic loss, Loss are the same characteristic.
    average_loss_train = eval_result_train["average_loss"]
    average_loss_test = eval_result_test["average_loss"]
    print("Average_loss: Train:{0} Test:{1}".format(average_loss_train**0.5,  average_loss_test**0.5))

    #prediction
    predict_results = model.predict(
        input_fn=tf.estimator.inputs.pandas_input_fn(x = predict_dataset[0], shuffle=False),
        predict_keys= None
    )

    return predict_results
```

LinearClassifier.py

```
import tensorflow_estimator as tf

def run_model(feature_columns, train_dataset, test_dataset, predict_dataset, model_dir ,
              optimizer=None, batch_size=None, num_epochs = None, steps = None,
              label_vocabulary = []):
    """
    create model
    """

    label_vocabulary = [str(x) for x in label_vocabulary]

    model = tf.estimator.LinearClassifier(
        feature_columns = feature_columns,
        optimizer = optimizer,
        label_vocabulary = label_vocabulary,
        n_classes = len(label_vocabulary),
        model_dir= model_dir, # none - used default

        config = tf.estimator.RunConfig(
            save_checkpoints_steps=500,  # Save checkpoints every x steps.
            keep_checkpoint_max=10000  # Retain the x most recent checkpoints.
        )
    )

    print("Train dataset length:",len(train_dataset[0]))

    #train model
    model.train(
        input_fn = tf.estimator.inputs.pandas_input_fn(
            x = train_dataset[0],
```

```
        y = train_dataset[1],
        shuffle = True,
        batch_size = batch_size,
        num_epochs = num_epochs), steps = steps)

from tensorflow.python.training import checkpoint_management
for ckpt_file in checkpoint_management.get_checkpoint_state(checkpoint_dir = model_dir).all_model_checkpoint_paths:
    eval_result_test = model.evaluate(input_fn=tf.estimator.inputs.pandas_input_fn(
        x=test_dataset[0],
        y=test_dataset[1],
        shuffle=False),
        name="Test", checkpoint_path=ckpt_file)

    eval_result_train = model.evaluate(input_fn=tf.estimator.inputs.pandas_input_fn(
        x=train_dataset[0],
        y=train_dataset[1],
        shuffle=False), name="Train", checkpoint_path=ckpt_file)

# Evaluate how the model performs on test dataset for last checkpoint.
eval_result_train = model.evaluate(input_fn = tf.estimator.inputs.pandas_input_fn(
    x = train_dataset[0],
    y = train_dataset[1],
    shuffle=False),
    name = "Train Final")

eval_result_test = model.evaluate(input_fn = tf.estimator.inputs.pandas_input_fn(
    x = test_dataset[0],
    y = test_dataset[1],
    shuffle=False),
    name = "Test Final")


# print loss. Average loss, Average quadratic loss, Loss are the same characteristic.
average_loss_train = eval_result_train["average_loss"]
average_loss_test = eval_result_test["average_loss"]
print("Average_loss: Train:{0} Test:{1}".format(average_loss_train**0.5,  average_loss_test**0.5))

#prediction
predict_results = model.predict(
    input_fn=tf.estimator.inputs.pandas_input_fn(x = predict_dataset[0], shuffle=False),
    predict_keys= None
)

return predict_results
```

DOKUMENTĀRĀ LAPA

Maģistra darbs "**Transportlīdzekļu apdrošināšanas atlīdzību prognozēšana izmantojot dziļos neironu tīklus"** izstrādāts LU Datorikas fakultātē.

Darba teksta galīgā versija izgatavota 16.05.2019.

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: **Imants Petrovs_____16.05.2019**

Ar savu parakstu apliecinu, ka esmu lasījis augstāk minēto maģistra darbu un atzīstu to par p i e m ē r o t u / n e p i e m ē r o t u (nevajadzīgo svītrot) aizstāvēšanai Latvijas Universitātes datorzinātņu maģistrantūrā.

Darba vadītāja: **Dr.dat. Guntis Bārzdiņš_____16.05.2019**

Darbs iesniegts **maģistratūras sekretariātā** 20.05.2019.
Ar šo es apliecinu, ka darba elektroniskā versija ir augšupielādēta LU informatīvajā sistēmā.

Studiju metodiķe: _____.

Recenzents: _____

Darbs aizstāvēts maģistra gala pārbaudījuma komisijas sēdē
_____ prot. Nr. _____

Komisijas sekretārs: _____