VILNIUS TECH
Vilniaus Gedimino technikos universitetas

Vilniaus Gedimino Technikos Universitetas
Fundamentinių mokslų fakultetas
Matematinės statistikos katedra

Mantas Urbonas

PYTHON BIBLIOTEKŲ TAIKYMAS KLIENTŲ NETEKIMO ANALIZEI TELEKOMUNIKACIJŲ SRITYJE

Application of Python libraries to churn analysis in telecommunications

Baigiamasis bakalauro darbas

Duomenų analizės technologijų studijų programa, valstybinis kodas 6121AX009
Statistikos studijų kryptis

Vilnius, 2023

Vilniaus Gedimino Technikos Universitetas
Fundamentinių mokslų fakultetas
Matematinės statistikos katedra

TVIRTINU
Katedros vedėjas

_____
(Parašas)

_____
(Vardas, pavardė)

_____
(Data)

Mantas Urbonas

PYTHON BIBLIOTEKŲ TAIKYMAS KLIENTŲ NETEKIMO ANALIZEI TELEKOMUNIKACIJŲ SRITYJE

Application of Python libraries to churn analysis in telecommunications

Baigiamasis bakalauro darbas

Duomenų analizės technologijų studijų programa, valstybinis kodas 6121AX009
Statistikos studijų kryptis

Vadovas _____ _____ _____
(Pedagoginis vardas, vardas, pavardė) (Parašas) (Data)

Vilnius, 2023

Turinys

1 Įvadas

Vykstant sparčiam ekonominiam augimui, stipriai vystosi ir daugelis įmonių – technologijų sektorius, telekomunikacijos tinklai, įvairios paslaugos, maisto tiekėjai ir t.t.. Dėl tokio spartaus augimo atsiranda vis didesnė konkurencija tarp įmonių, sukuriant platų siūlomų paslaugų spektrą ir didinant kliento pasirinkimo laisvę. Dėl didėjančio pasirinkimo tampa paprasčiau pakeisti nepatikusias prekes, paslaugas, įmones ar prekės ženklą, siekiant daugiau naudos sau, kaip klientui. Toks lengvas paslaugų keitimas sudaro sudėtingas sąlygas verslams, kurie gali patirti didelius nuostolius, ypač prarandant lojalių klientų bazę. Nepalankų klientų apsisprendimą gali nulemti įvairūs įmonės veiksniai kaip: kainų kėlimai, suprastėjusi produktų kokybė, prastas aptarnavimas ar patrauklesnės sąlygos svetur. Dėl tokių veiksmų įmonės dažnai taiko įvairius metodus, siekdamos išlaikyti klientą, palaikyti pinigų srautą ir išlikti konkurencingos dabartinėje rinkoje. Dažniausiai taikomi metodai – rizikos analizė, arti surišta su Churn (klientų netekimo) analize, konkurentų analizė ir t.t..

Kadangi daugelis įmonių keliasi į internetinę erdvę, visas paslaugas labai lengva lyginti su konkurentais ir taip atlikti finansinius sprendimus. Viskam vykstant internetinėje erdvėje galima surinkti labai daug vertingos informacijos apie klientus ir jų polinkį pasirinkti ir atsisakyti paslaugų bei nustatyti šį sprendimą lemenčius veiksnius, taip lengviau pasirenkant tinkamus verslo sprendimus. Vienas iš dažnai taikomų verslo sprendimų - klientų segmentacija, t.y., paslaugų gavėjų suskirstymas į įvairias grupes. Identifikavus lojaliausias ir pelningiausias paslaugų gavėjų grupes, vykdoma detalesnė analizė, kurios metu nustatomi specifiniai grupės poreikiai. Tokiu būdu tikslingai parenkami resursai, skirti klientus pritraukti bei juos išlaikyti. Kadangi duomenų rinkiniai internetinėje erdvėje yra labai platūs, todėl taikant paprastesnius metodus gali būti sudėtinga, siekiant identifikuoti pagrindines savybes. Tačiau taikant mašininį mokymąsi (mašininio mokymosis metodus) nesudėtinga atrinkti svarbiausias klientų charakteristikas bei pagal jas nustatyti tendencijas, kurias gali būti sudėtinga įžvelgti.

Šio darbo metu buvo atlikta telekomunikacijų sektoriaus Churn (klientų netekimo) analizė, kuri fokusuojasi į klientų išlaikymą bei jų netekimo prevenciją. Klientų netekimas yra aktuali šiuolaikio verslo problema. Daug pastangų yra sutelkiama siekiant klientą išlaikyti, kadangi netekus klientų, reikalingi didesni resursai naujiem klientams pritraukti.

2 Baigiamojo darbo aprašas

2.1 Baigiamojo darbo tikslas

Pagrindinis tikslas - pasirinktam žymėtų duomenų rinkiniui pritaikyti Python bibliotekas, skirtas klasifikavimui, reikšmingų atributų nustatymui ir kitokiai analizei.

2.2 Baigiamojo darbo uždaviniai

1. Apžvelgti laisvai prieinamus duomenų rinkinius ir pritaikyti vieną Churn analizei.
2. Apžvelgti metodus skirtus tokio tipo duomenų analizei ir aktualių uždavinių sprendimui.
3. Suformuluoti prognozės uždavinį kaip klasifikavimo uždavinį ir su pasirinktu metodu jį išspręsti.
4. Išanalizuoti gautus rezultatus: tikslumo metrikų skaičiavimas, reikšmingų atributų nustatymas ir kt..

2.3 Churn analizė

Klientų netekimo analizė (Churn) svarbi telekomunikacijų sektoriuje, nes padeda įmonėms suprasti, kodėl klientai atsisako paslaugų, ir nustatyti modelius bei tendencijas, kuriais remiantis galima parengti klientų išlaikymo ir netekimo mažinimo strategijas. Analizuodamos duomenis apie klientų elgseną, demografinius duomenis, naudojimo modelius ir įvairius rodiklius, įmonės gali nustatyti, kuriems klientams gresia paslaugų atsisakymo pavojus, ir rengti tikslines kampanijas bei paskatas, kad juos išlaikytų. Be to, klientų skaičiaus mažėjimo analizė gali padėti bendrovėms suprasti, kurie klientų tipai yra vertingiausi ir pelningiausi, ir parengti strategijas, kaip pritraukti ir išlaikyti daugiau tokių klientų. Norint išlaikyti stabilią ir pelningą klientų bazę, telekomunikacijų bendrovėms labai svarbu suprasti ir valdyti klientų skaičiaus mažėjimą. Be to, sumažinus klientų Churn'ą, įmonė gali gauti didelės finansinės naudos, nes naujų klientų įsigijimas gali būti brangesnis nei esamų klientų išlaikymas [1].

2.4 Churn problemos

- Viena iš pagrindinių problemų, susijusių su Churn (klientų netekimo) analize, yra ta, kad gali būti sunku tiksliai numatyti, kurie klientai gali atsisakyti paslaugų. Taip yra todėl, kad gali būti daug įvairių veiksnių, kurie lemia kliento sprendimą nutraukti paslaugą, ir šie veiksniai gali skirtis priklausomai nuo kliento. Be to, klientų elgesys ir pageidavimai laikui bėgant gali keistis, todėl gali būti sunku nustatyti modelius ir tendencijas, rodančias paslaugų atsisakymo tikimybę.

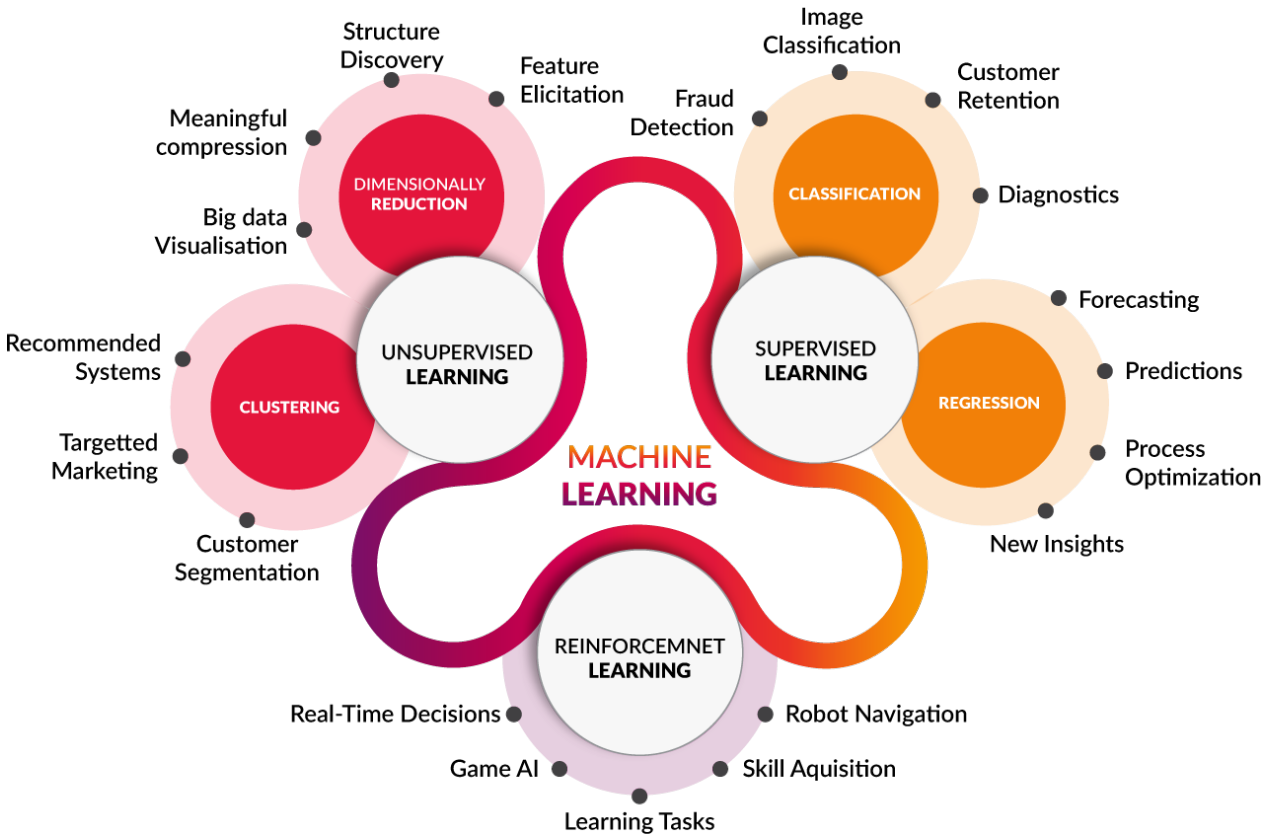
- Kita problema, susijusi su Churn analize, yra ta, kad gali būti sunku nustatyti klientų netekimo priežastis, ypač tas, į kurias lengvai galime atsižvelgti. Taip yra todėl, kad klientai gali nurodyti įvairias paslaugų atsisakymo priežastis, todėl gali būti sunku nustatyti, kurie veiksniai yra svarbiausi priimant sprendimą. Nežinant pagrindinių priežasčių, sunku nustatyti tikrąją problemą ir parengti veiksmingą sprendimą, leidžiantį pristabdyti klientų netekimą.

Apibendrinant galima teigti, kad Churn analizė yra sudėtinga užduotis dėl dinamiško ir daugiafaktorinio klientų sprendimų priėmimo pobūdžio, todėl norint gauti aukšto patikimumo prognozes, reikia atidžiai apsvarstyti kokius duomenis naudoti bei kokius analizės metodus taikyti.

3 Mašininis mokymasis

Kompiuterio mokymas [11], dar kitaip vadinamas kaip mašininis mokymasis yra dirbtinio intelekto metodų klasė, kuriai būdingas ne tiesioginis problemos sprendimas, o mokymasis, kaip pritaikyti daugelio panašių problemų sprendimus (1 pav.). Tai yra programų kūrimo būdas, kai sukurta sistema prisitaiko prie duomenų t.y. apsimoko. Šie algoritmai dažniausiai sugeba spręsti sudėtingesnes problemas ir ilgainiui pasiekti geresnių rezultatų, kaupdami patyrimą [8].

Mašininis mokymasis gali būti galingas įrankis analizuojant klientų netekimą, nes gali padėti įmonėms nustatyti klientų elgesio modelius ir tendencijas, kurios gali rodyti klientų netekimo tikimybę. Churn'o analizė dažniausiai orientuojasi į klientų klasifikavimo problemą ir būtent jai spręsti yra naudojami įvairūs mašininio mokymosi algoritmai. Dažniausiai naudojami mašininio mokymosi algoritmai, kuriuos galima taikyti klientų netekimo analizei – sprendimų medžiai (angl. *Decision trees*), atsitiktiniai miškai (angl. *Random forest*), gradientinį didinimą (angl. *Gradient boosting*) ir neuroninius tinklus.



1 pav.: Mašininis mokymasis (Šaltinis: https://www.pngaaa.com/detail/3730478)

3.1 Prižiūrimas mašininis mokymasis

Churn prognozės uždaviniui spręsti paprastai naudojamas prižiūrimas mašininis mokymasis (angl. *Supervised Machine learning*). Prižiūrimas mašininis mokymasis – tai mašininio mokymosi rūšis, kai modelis mokomas pagal pažymėtus duomenis, t.y., duomenis sudaro įvesties požymiai ir atitinkamos išvesties žymos. Prižiūrimo mokymosi tikslas – išmokyti modelį, kuris galėtų tiksliai prognozuoti naujus, nematytus duomenis. Labiausiai paplitęs prižiūrimo mokymosi tipas yra klasifikavimas, kai išvesties etiketės yra diskrečios kategorijos, ir regresija, kai išvesties etiketės yra tolydžios reikšmės.

Norint apmokyti modelį, prižiūrimam mokymuisi reikia sužymėto duomenų rinkinio. Modeliui apmokyti naudojamas pažymėtų duomenų rinkinys, vadinamas mokymo duomenų rinkiniu, o tikslumas tikrinamas naudojant atskirą duomenų rinkinį, vadinamą testavimo duomenų rinkiniu. Modelis apmokomas apibendrinti nematytą testavimo duomenų rinkinį.

Vienas iš įprastų mašininio mokymosi metodų, taikomų klientų netekimo analizei (Churn), yra klasifikavimo uždavinio sprendimas, pagal kurį sukuriamas kliento elegsį progozuojantis mode-

lis. Modeliui reikalingi įvesties duomenys yra klientą apibūdinančių požymių rinkinys, pavyzdžiui, jo demografiniai duomenys, vartojimo įpročiai ir ankstesnė sąveika su įmone. Duomenyse yra sužymima, kurie klientai atsisakė paslaugų, o kurie iki dabar jomis naudojasi. Pasitelkiant šiuos duomenis, apskaičiuojama kliento turimų paslaugų atsisakymo tikimybė. Gavus prognozę, kad klientas yra linkęs atsisakyti paslaugos, galima imtis prevencijos priemonių, siekiant jį išlaikyti.

Yra daug įvairių algoritmų ir metodų, kuriuos galima naudoti prižiūrimam mokymuisi, įskaitant sprendimų medžius (angl. *Decision Trees*), atsitiktinius miškus (angl. *Random Forest*), k artimiausių kaimynų (angl. *k-nearest neighbors*), tiesinę regresiją (angl. *linear regression*), logistinę regresiją (angl. *logistical regression*) ir atraminių vektorių mašinas (angl. *Support Vector Machines*). Algoritmo pasirinkimas priklauso nuo konkrečios problemos ir duomenų savybių.

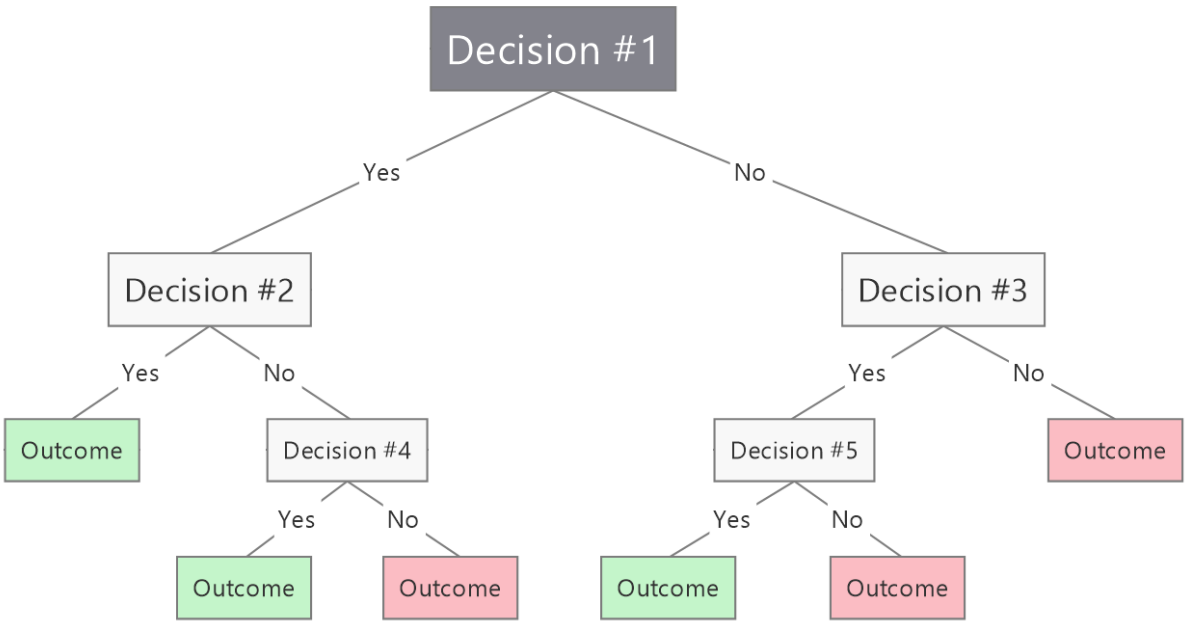
Alternatyvus metodas – neprižiūrimas mašininis mokymąsi, pavyzdžiui, klasterizavimas, taikomas segmentuoti klientus pagal jų elgseną ir rasti dėsningumus, kurie gali padėti nustatyti didelės rizikos klientus.

Mašininis mokymasis gali būti labai naudingas atliekant klientų netekimo analizę, nes padeda automatizuoti didelio duomenų kiekio analizės procesą, nustatyti sudėtingus dėsningumus bei gan tiksliai prognozuoti klientų elgseną. Apmokyti modeliai, pagal istorinius klientų duomenis, modeliai gali numatyti, kuris klientas greičiausiai atsisakys savo paslaugų, o įmonės gali kurti tikslines išlaikymo priemones, kad išsaugotų šiuos klientus.

3.2 Sprendimų medžiai

Sprendimų medžiai yra populiarus įrankis, skirtas prognoziniams modeliams kurti mašininio mokymosi ir duomenų gavybos srityse. Jie plačiai naudojami įvairiose srityse, pavyzdžiui, klasifikavimui, regresijai ir požymių atrankai[3]. Sprendimų medis – tai į medį panašus modelis, vaizduojantis sprendimų seką ir galimas jų pasekmes. Kiekvienas medžio mazgas reiškia įvesties kintamojo sprendimą arba testą, o kiekviena šaka – testo rezultatą. Medžio lapai reiškia galutinį sprendimą arba prognozę.

Sprendimų medžius lengva interpretuoti ir vizualizuoti, todėl jie naudingi siekiant suprasti sprendimų priėmimo procesą (2 pav.) ir pateikti modelio rezultatus ne specialistams[4]. Jie gali apdoroti tiek kategorinius, tiek skaitinius duomenis ir gali automatiškai atlikti požymių atranką, nustatydami informatyviausius kintamuosius. Be to, sprendimų medžius galima lengvai integruoti į kitus metodus, tokius kaip atsitiktinis miškas ir gradientinis didinimas, kurie gali padidinti modelio tikslumą ir sumažinti persimokymą[5].



2 pav.: Sprendimų medis

Pagrindinė sprendimų medžių idėja – medžiai kuriami naudojant rekursyvų skaidymo procesą, kai kiekviename medžio mazge duomenys padalijami į du poaibius pagal pasirinktą skaidymo tašką. Skirstymo taškas pasirenkamas taip, kad būtų sumažinta gautų poaibių priemaiša, kuri pa-

prastai matuojama naudojant tokius rodiklius kaip Gini priemaiša (angl. *Gini impurity*) arba entropija/informacijos prieaugį (angl. *entropy/information gain*). Algoritmo tikslas – minimizuoti Gini indeksą arbą entropiją. Standartiškai naudojamas Gini indeksas, nes skaičiavimai yra paprastesni, o abu kriterijai skaičiuoja padalijimų tyrumą. Tikslas padalinti duomenų rinkinį taip, kad geriausiai atsiskirtų tiriamoji savybė.

$$Gini = 1 - \sum_{j}(p_j^2) \tag{1}$$

$$Entropy = \sum_{j} -(p_j * log_2(p_j)) \tag{2}$$

,kur p_j yra tikimybė patekti į pasirinktą klasę.

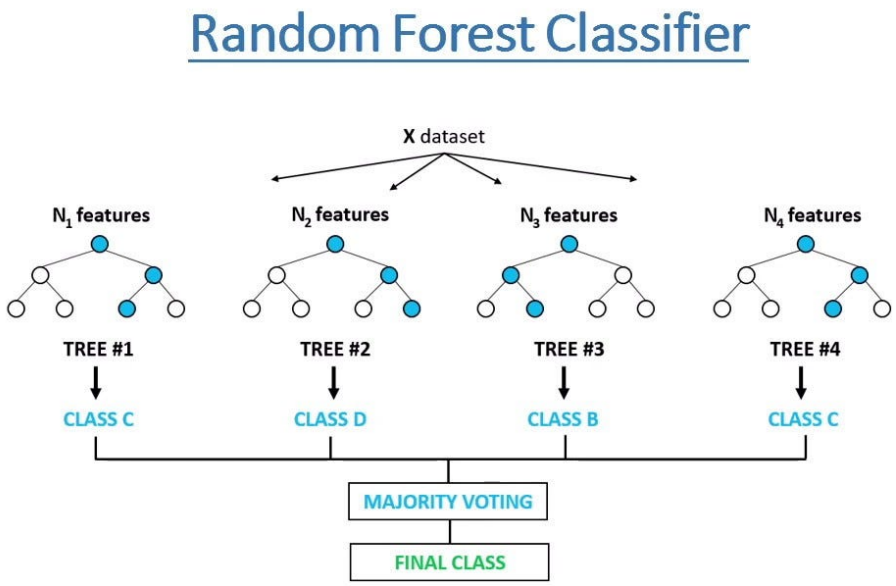
Tačiau sprendimų medžiai yra linkę į persimokymą, ypač kai medis yra gilus ir sudėtingas. Šiai problemai sušvelninti gali būti taikomi įvairūs apribojimai ir normalizacijos metodai. Be to, sprendimų medžiai yra jautrūs nedideliems įvesties duomenų pokyčiams ir gali sukurti skirtingus medžius skirtingoms duomenų imtims. Šiai problemai spręsti galima taikyti anksčiau paminėtus metodus ir kryžminio tikrinimo technikas, kurie pagerina modelio stabilumą ir apibendrinimo rezultatus.

3.3 Atsitiktinio miško klasifikatorius

Atsitiktiniai miškai (angl. *Random Forest*) yra mašininio mokymosi algoritmas, plačiai taikomas klasifikavimo ir regresijos užduotims atlikti. Tai mokymosi metodas, kuris sujungia kelis sprendimų medžius, kad būtų sukurtas patikimesnis ir tikslesnis modelis. Atsitiktiniai miškai yra lengvai plečiami ir gali apdoroti didelius duomenų rinkinius, pasižyminčius dideliu dimensiškumu.

Pagrindinė atsitiktinio miško idėja – sukurti daug sprendimų medžių (dėl to vadinamas miškas (žr. 3 pav.)), kurių kiekvienas remiasi atsitiktiniu įvesties požymių poaibiu ir atsitiktiniu mokymo duomenų poaibiu. Dėl atsitiktinių požymių ir duomenų poibių šis metodas tampa atsparesnis persimokymui ir mažiau prisitaiko prie duomenų ir gali atlikti tikslesnes predikcijas, nei vienas individualus sprendimų medis.

Sukūrus medžius, atsitiktinio miško algoritmas sujungia jų prognozes, kad gautų galutinį rezultatą. Sprendžiant klasifikavimo problemas, išvestis paprastai yra atskirų medžių išvesties balsų dauguma, o sprendžiant regresijos problemas, išvestis paprastai yra atskirų medžių išvesties vidurkis arba mediana.



3 pav.: Atsitiktinio miško balsavimas (Šaltinis: https://medium.com/analytics-vidhya/random-forest-classifier-and-its-hyperparameters-8467bec755f6

Vienas iš pagrindinių atsitiktinių miškų privalumų yra jų gebėjimas apdoroti didelės dimensijos duomenis ir išvengti persimokymo, kuris gali įvykti, kai naudojamas vienas sprendimų medis. Atsitiktinai parenkant požymių ir mokymo duomenų poaibius, kiekvienas medis priverstas sutelkti dėmesį į skirtingus duomenų aspektus, o tai padeda sumažinti dispersiją ir pagerinti apibendrinimo rezultatus.

Atsitiktiniai miškai sėkmingai taikomi įvairiose srityse, įskaitant teksto klasifikavimą, vaizdų klasifikavimą ir genų raiškos analizę. Jie taip pat yra populiarus pasirinkimas požymių atrankos ir duomenų tyrinėjimo užduotims, nes leidžia suprasti santykinę skirtingų įvesties požymių svarbą.

Atsitiktinio miško modelio mokymo procesas paprastai apima hiperparametrų, tokių kaip medžių skaičius, didžiausias kiekvieno medžio gylis ir atsitiktinių požymių poaibių dydis, koregavimą. Modelį galima įvertinti naudojant tokius rodiklius kaip tikslumas (angl. *accuracy*), preciziškumas (angl. *precision*), jautrumas (angl. *recall*) ir F1-balas.

Apibendrinant galima teigti, kad atsitiktinis miškas yra galingas ir lankstus algoritmas, kurį galima naudoti ir klasifikavimo, ir regresijos uždaviniams spręsti, ir kuris ypač gerai tinka didelės apimties duomenims. Dėl gebėjimo išvengti persimokymo ir suteikti įžvalgų apie požymių svarbą šis algoritmas yra populiarus pasirinkimas įvairiose srityse.

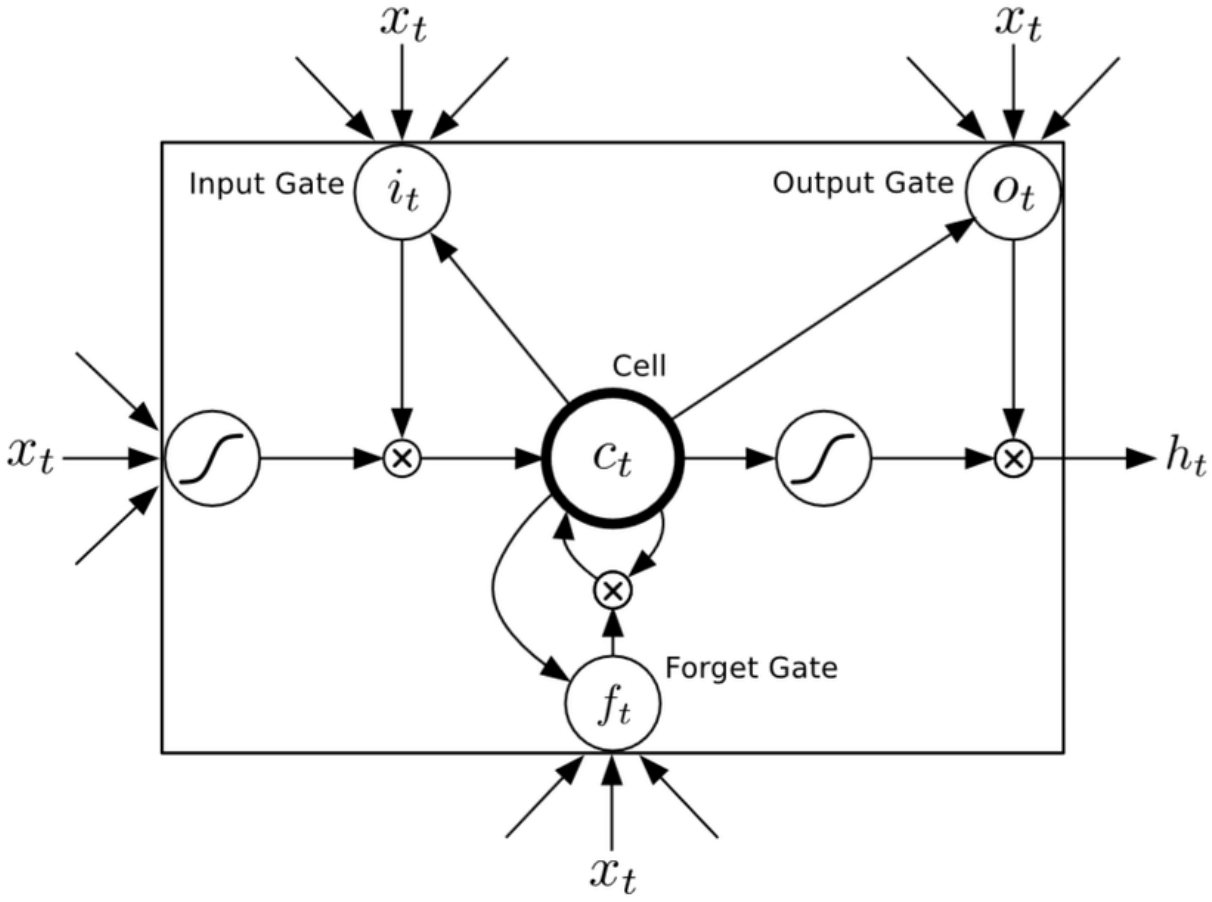
3.4 LSTM neuroninis tinklas

Ilgalaikė trumpalaikė atmintis (angl. *long short-term memory (LSTM)*) yra rekurentinių neuroninių tinklų (angl. *Recurrent neural network(RNN)*) architektūros tipas, plačiai naudojamas natūralios kalbos apdorojimo (angl. *Natural Language Processing (NLP)*), kalbos atpažinimo ir kituose "sekos į seką" prognozavimo uždaviniuose. LSTM tinklai sukurti, kad išspręstų standartiniuose RNN tinkluose kylančią nykstančio gradiento problemą, dėl kurios tinklui sunku išmokti ilgalaikių priklausomybių nuosekliuose duomenyse.

Pagrindinė LSTM idėja – įdiegti atminties ląsteles, kurios gali saugoti informaciją ilgą laiką, ir vartus, kurie valdo informacijos srautą į ląsteles ir iš jų. Atminties ląstelės atnaujinamos kiekvienu laiko žingsniu, remiantis įvestimi, ankstesne paslėpta būsena ir ankstesne atminties būsena. Vartai yra atsakingi už informacijos srauto į atminties ląsteles ir iš jų reguliavimą, todėl tinklas gali antrankiai įsiminti arba pamiršti informaciją.

Pagrindiniai LSTM tinklo komponentai yra įėjimo vartai, užmiršimo vartai, išėjimo vartai

ir atminties ląstelė (žr. 4 pav.). Įvesties vartai lemia, kiek naujos informacijos pridedama į atminties ląstelę, užmiršimo vartai – kiek senos informacijos išmetama, o išvesties vartai – kiek informacijos nuskaitoma iš atminties ląstelės. Pati atminties ląstelė yra atsakinga už informacijos saugojimą ir atnaujinimą.



4 pav.: LSTM architektūra (Šaltinis: Graves, Alex. "Generating sequences with recurrent neural networks." arXiv preprint arXiv:1308.0850 (2013).

LSTM tinklo mokymo procesas apima grįžtamąją sklaidą per laiką (angl. *backpropagation through time (BPTT)*), kai nuostolių funkcijos gradientas tinklo parametrų atžvilgiu apskaičiuojamas kiekvienu laiko žingsniu ir skleidžiamas atgal per visą seką. Tada parametrai atnaujinami taikant optimizavimo algoritmą, pavyzdžiui, stochastinį gradientinį nusileidimą (angl. *stochastic gradient descent (SGD)*) arba Adam'o algoritmą.

LSTM yra galingas įrankis gebantis spręsti daugelį "sekos į seką" prognozavimo uždavinių, įskaitant mašininį vertimą, kalbos ir rašysenos atpažinimą. Tačiau vis dar yra daug iššūkių ir atvirų

klausimų, susijusių su LSTM ir kitų RNN architektūrų naudojimu, pavyzdžiui, kaip veiksmingai tvarkyti ilgas sekas, ir kaip tvarkyti triukšmingus ar neišsamius duomenis.

4 Churn uždavinio matematinis formulavimas

Vienas iš būdų matematiškai suformuluoti Churn uždavinį – naudoti binarinį klasifikavimą. Taikant šį metodą, kiekvienas klientas, atsižvelgiant į jo elgseną, klasifikuojamas kaip "Churner"(atsisakantis paslaugų) arba "Non-Churner"(pasiliekantis). Tikslas - sukurti modelį, pagal kurį būtų galima tiksliai numatyti, ar klientas atsisakys paslaugų, ar ne.

Binarinio klasifikavimo uždavinį galima formuluoti taip: Tegul X yra iš duomenų rinkinio gautas vektorius, kuriame kiekvienas elementas x_i aprašo kliento požymius, pavyzdžiui, kliento demografinius arba naudojimo duomenis. Iš viso klientų yra N. Tegul Y yra tikslinis vektorius, kurio kiekvienas elementas yra 0 (pasilieka) arba 1 (išeina). $\bar{Y}$ yra prognozuojamų atsakymų vektorius. Tai galima užrašyti matematine išraiška:

$$\bar{Y}(X,H) = \left[\bar{y}_i(x_i) = \begin{cases} 1, & \text{jei churn}(x_i,H) \\ 0, & \text{kitu atveju} \end{cases}, \quad i \in [1,\ldots,N] \right] \tag{3}$$

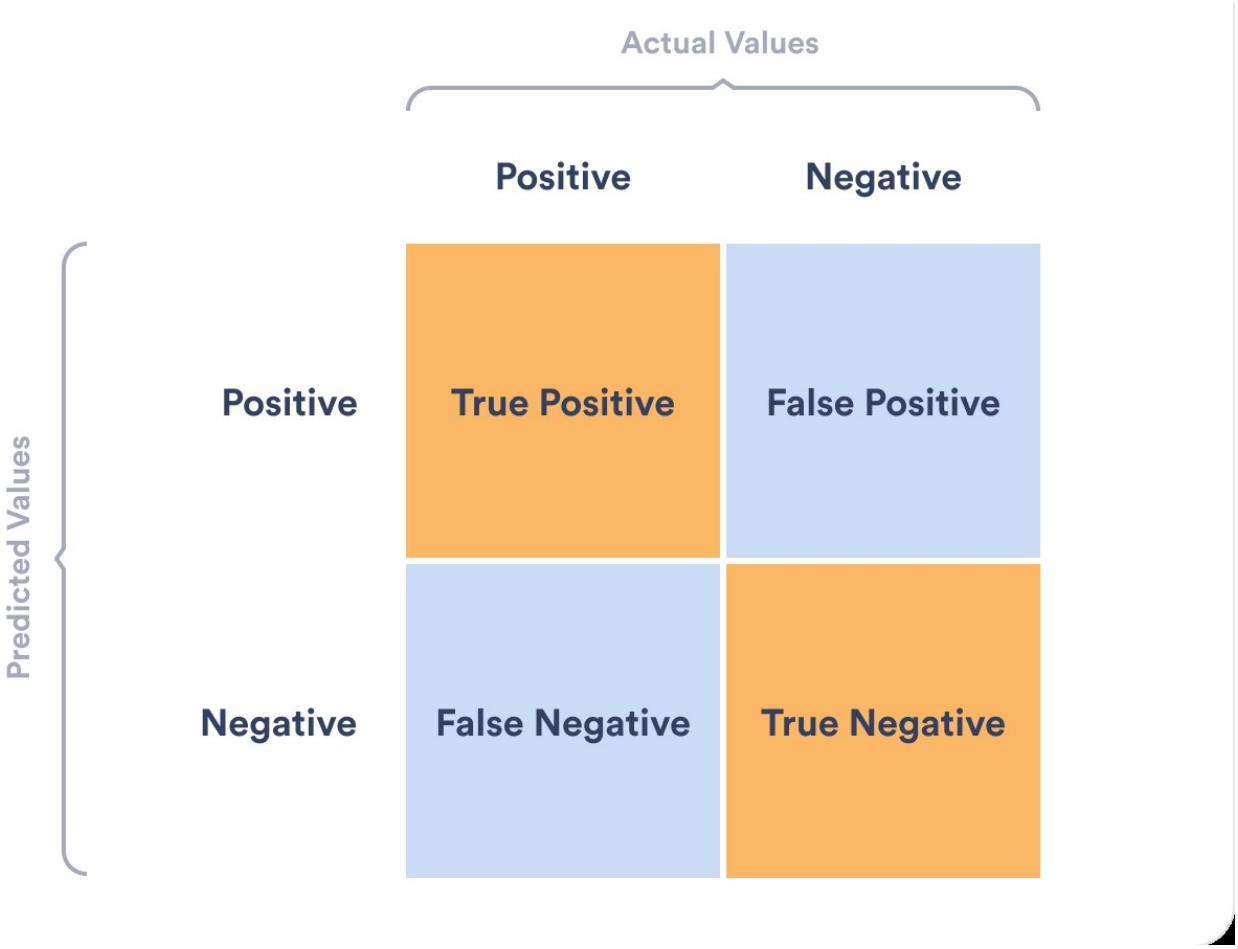
Čia H bendru atveju yra parametrų aibė, o Churn atveju – pasirinkto mašininio mokymosi metodu gauta klasifikavimo modelio funkcija. Uždavinio tikslas yra minimizuoti daromas paklaidas (M), pagal pasirinktą parametrą H, tai galima užrašyti kaip minimizavimo uždavinį:

$$\min_H M(\bar{Y}(X,H),Y) \tag{4}$$

Yra labai daug parametrų, pagal kuriuos galima vertinti algoritmo efektyvumą.

4.1 Churn uždavinio sprendimo sėkmingumo vertinimas: efektyvumo rodikliai

Yra keletas efektyvumo rodiklių, kuriuos galima naudoti norint įvertinti mašininio mokymosi rezultato prognozavimo modelio efektyvumą. Binariniam klasifikavimui dažnai naudojama maišos matrica (žr. pav. 5), parodanti, kokias predikcijas atlieka modelis, ir kokią dalį atspėja teisingai.



5 pav.: Maišos matrica

Kai kurios iš dažniausiai naudojamų efektyvumo metrikų klasifikavimo problemoms, tokioms kaip Churn analizė, spręsti yra šios:

1. Tikslumas (angl. *Accuracy*): Šis rodiklis vertina teisingai suklasifikuotų atvejų (ir teigiamų, ir neigiamų) dalį iš visų atvejų. Tai paprastas ir intuityvus matas, tačiau jis gali būti klaidinantis, jei duomenų rinkiniai yra nesubalansuoti (kai tikslinės klasės atstovaujamos nevienodai).

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{5}$$

2. Preciziškumas (angl. *Precision*): Šis rodiklis parodo teigiamų prognozių dalį iš visų teigiamų prognozių. Ji naudinga tais atvejais, kai klaidingai teigiami rezultatai brangiai kainuoja ir norima įsitikinti, kad modelis nepateikia per daug klaidingai teigiamų prognozių.

$$Precision = \frac{TP}{TP+FP} \tag{6}$$

3. Jautrumas (angl. *Recall arba True positive rate (TPR)*): Šis rodiklis parodo teisingų teigiamų prognozių dalį iš visų faktinių teigiamų atvejų. Šis rodiklis naudingas tais atvejais, kai klaidingai neigiami rezultatai brangiai kainuoja ir norima įsitikinti, kad modelis nepateikia per daug teigiamų atvejų.

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

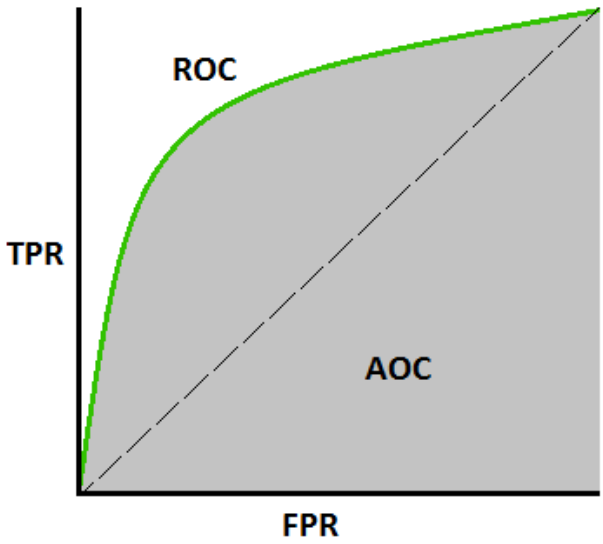
4. Klaidingų teigiamų rezultatų skaičius (angl. *False positive rate (FPR)*): Šis rodiklis parodo neigiamų atvejų, kurie buvo klaidingai klasifikuoti kaip teigiami, dalį iš visų faktinių neigiamų atvejų. Jis leidžia suprasti modelio gebėjimą sumažinti klaidingai teigiamų atvejų skaičių, o tai ypač svarbu atliekant klientų skaičiaus netekimo analizę, siekiama išvengti klaidingo klientų, kurie atsiake paslaugų, identifikavimo.

$$FPR = \frac{FP}{FP + TN} \tag{8}$$

5. F1-balas(angl. $F1 - score$): tai tikslumo ir atšaukimo harmoninis vidurkis, pusiausvyros metrika.

$$F1\text{-}score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{9}$$

6. AUC-ROC kreivė: Tai grafikas, kuris iliustruoja dvejetainio klasifikatoriaus sistemos našumą keičiant jos diskriminavimo ribą (žr. 6 pav.). Plotas po ROC kreive (AUC) gali būti naudojamas kaip dvejetainio klasifikavimo problemos našumo matas.



6 pav.: AUC-ROC kreivė

7. G-vidurkis: tai yra pusiausvyros tarp tikrojo teigiamo ir specifiškumo rodiklis. Jis naudingas, kai yra nesubalansuotas duomenų rinkinys.

$$G = \sqrt{\frac{TP}{TP+FN} * \frac{TN}{TN+FP}} \tag{10}$$

Taip pat svarbu pažymėti, kad šie rodikliai nėra viena kitą pakeičiančios ir jos naudojamos modelio našumui įvertinti, pateikiant skirtingus požiūrius į tai, kaip gerai modeliui sekasi prognozuoti. Priklausomai nuo konkretaus naudojimo atvejo, skirtingi rodikliai gali būti svarbesni už kitus, pavyzdžiui, kai vienos iš klasės atstovų yra labai mažai, tai tikslumo rodikliai rodys gerą rezultatą, net kai metodas nesurado nei vieno mažumos atstovo, tačiau kartais tokie atstovai yra patys svarbiausi, nes tarkim gali būti kenkėjiškų vartotojų mažuma. Būtent toks atvejis ir bus nagrinėjamas šiame darbe, kai mažoji dalis imties turi didelę įtaką visam uždavinio sprendimui. F1-balas ir jautrumas tampa svarbūs rodikliai, vertinant modelio patikimumą. Bendrai, kai duomenų rinkinyje gan smarkiai skiriasi skirtingų klasių atstovų proporcijos, vadinami – išbalansuotieji, kuriuos nagrinėsime vėliau.

4.2 Churn uždavinio sprendimo sėkmingumo vertinimas: reikšmingumo rodikliai

Reikšmingumo rodikliai – vertinga mašininio mokymosi sąvoka, padedanti suprasti santykinį įvesties požymių indėlį į tikslių prognozių sudarymą. Analizuodami požymių svarbą galime nustatyti didžiausią įtaką turinčius požymius ir suprasti pagrindinius požymių ir tikslinio kintamojo ryšius. Požymių svarbą galima apskaičiuoti taikant įvairius metodus, pavyzdžiui, Gini svarbą atsitiktiniuose miškuose arba koeficientus tiesiniuose modeliuose. Šie svarbos balai leidžia nustatyti požymių prioritetus tolesnei analizei, požymių atrankai arba modelio optimizavimui. Požymių svarbos supratimas ne tik pagerina mūsų supratimą apie duomenis, bet ir padeda kurti aiškesnius ir veiksmingesnius mašininio mokymosi modelius.

Darbe bus nagrinėjami atsitiktinių miškų algoritmo reikšmingumo rodikliai, kurie yra sudaromi ir interpretuojami tokia tvarka:

1. Atsitiktinio miško mokymas: algoritmas sukuria sprendimų medžių skaičių, naudodamas metodą, vadinamą "bootstrap" arba "bagging". Kiekvienas sprendimų medis mokomas pagal atsitiktinai atrinktą mokymo duomenų poaibį ir atsitiktinai parinktą įvesties požymių poaibį.

2. Gini svarba: matuoja bendrą priemaišų sumažėjimą, pasiektą skaidant pagal tam tikrą požymį visuose atsitiktinio miško sprendimų medžiuose. Ji kiekybiškai parodo, kiek požymis pagerina tikslinio kintamojo homogeniškumą poaibiuose, sukurtuose atlikus skirstymus.

3. Skaičiavimo procesas: apskaičiuojant požymio svarbą pagal Gini reikšmę, atsitiktinių imčių miško algoritmas apibendrina kiekvieno miško sprendimų medžio atskirus svarbos balus. Požymio svarba apskaičiuojama kaip to požymio priemaišų sumažinimo, pasiekto visuose medžiuose, vidurkis arba svertinis vidurkis. Funkcijos, kurios nuolat prisideda prie priemaišų mažinimo ir tikslesnio skirstymo, paprastai yra svarbesnės.

4. Santykinė svarba: atsitiktinio miško algoritmo gautas požymių reikšmingumas paprastai normalizuojamas taip, kad jo suma būtų lygi 1, arba išreiškiamas procentais. Toks normalizavimas leidžia geriau interpretuoti ir palyginti skirtingų požymių svarbos vertes.

5. Svarbos aiškinimas: didesnės požymių svarbos reikšmės rodo, kad konkretus požymis turi didesnę įtaką atsitiktinio miško modelio prognozėms. Kita vertus, mažesnės svarbos reikšmės rodo, kad požymis turi mažesnę įtaką arba gali būti nesvarbus prognozavimui.

5 Įgyvendinimo dalis

5.1 Įgyvendinimo įrankiai

Baigiamajame bakalauro darbe programiniam Python kodui rašyti buvo naudojamasi viešai prieinama Google Colab Notebook aplinka. Duomenų apdorojimui naudota *pandas* biblioteka, duomenų vizualizavimui – *seaborn*, $matplotlib$ bibliotekos, mašininiam mokymuisi realizuoti naudotos *sklearn* bei $tensorflow.keras$ bibliotekos.

5.2 Pradinė duomenų analizė

Pradinė duomenų analizė (angl. *Exploratory data analysis*), toliau EDA yra procesas, kurio metu tyrinėjamas duomenų rinkinys siekiant jį geriau suprasti, aptikti anomalijas, pašalinti triukšmą ir parengti duomenis tolimesnei analizei arba modeliavimui. EDA apima įvairius metodus, tokius kaip duomenų vaizdavimas grafikais, statistinė analizė, išskirčių ir klaidų nustatymas bei duomenų normalizavimas.

Analizės tikslas yra atskleisti duomenų rinkinio charakteristikas, įskaitant vidurkį, dispersiją, pasiskirstymą ir ryšius tarp kintamųjų. EDA yra svarbi dėl to, kad padeda suprasti duomenų rinkinio svarbiausius atributus ir jų poveikį. EDA apima statistinės analizės ir vizualizacijos metodų naudojimą, kad būtų galima atskleisti duomenų rinkinio turinį ir struktūrą. Tai leidžia identifikuoti potencialias problemų sritis ir nustatyti, kokie yra svarbiausi kintamieji.

5.2.1 Duomenys

Duomenys yra gauti iš internetinio puslapio
$https://github.com/jamesrawlins1000/Telecom-CDR-Dataset-/blob/master/Call\%20Details-Data.csv$

Duomenų rinkinyje yra 101 174 unikalių eilučių(klientų)
Kiekvieno kliento duomenys aprašomi 17 remiantis savybių (žr. 7 pav.):

1. Telefono numeris
2. Kiek laiko klientas naudojasi tiekėjo paslaugomis
3. Garso pranešimų žinučių skaičius

4. Dienos/vakaro/nakties skambučiai (skambučių kiekis ir prakalbėtos minutės)

5. Dienos/vakaro/nakties mokęsčiai už skambučius

6. Tarptautiniai skambučiai (skambučių kiekis ir prakalbėtos minutės)

7. Mokęsčiai už tarptautinius skambučius

8. Skambučiai į klientų aptarnavimo centrus (skambučių kiekis)

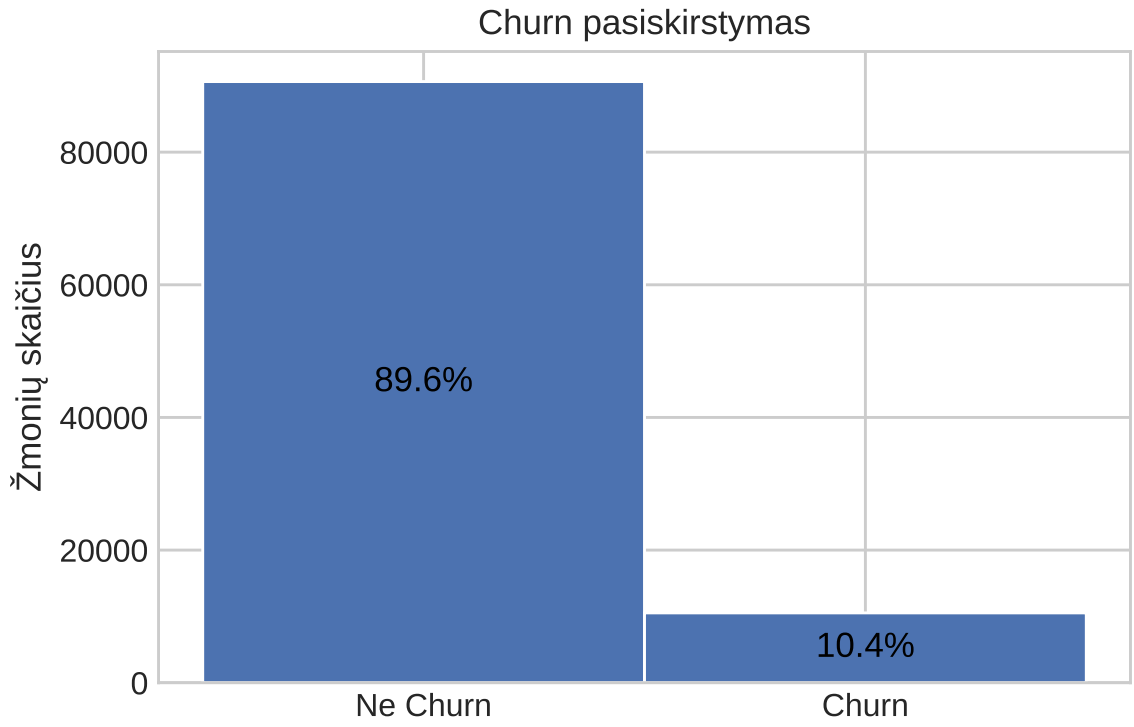
9. Klientų Churn (Taip/Ne)

	Phone Number	Account Length	VMail Message	Day Mins	Day Calls	Day Charge	Eve Mins	Eve Calls	Eve Charge	Night Mins	Night Calls	Night Charge	Intl Mins	Intl Calls	Intl Charge	CustServ Calls	Churn
0	382-4657	128	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.01	10.0	3	2.70	1	False
1	371-7191	107	26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.45	13.7	3	3.70	1	False
2	358-1921	137	0	243.4	114	41.38	121.2	110	10.30	162.6	104	7.32	12.2	5	3.29	0	False
3	375-9999	84	0	299.4	71	50.90	61.9	88	5.26	196.9	89	8.86	6.6	7	1.78	2	False
4	330-6626	75	0	166.7	113	28.34	148.3	122	12.61	186.9	121	8.41	10.1	3	2.73	3	False

7 pav.: Duomenų formatas

5.2.2 Churno pasiskirstymas

Norint korektiškai atlikti Churn'o analizę, duomenyse turi būti pakankamas Churn'erių kiekis, kad galėtume tiksliau identifikuoti pagrindines charakteristikas, kodėl žmonės atsisako paslaugų. Optimalus Churn'erių kiekis, reikalingas analizei atlikti, priklauso nuo kiekvienos specifinės situacijos ir reikmės. Tačiau dažniausiai optimalus kiekis yra pasirenkamas vadovaujantis verslo tikslais ir konkurencingu pranašumu rinkoje, pvz., jei rinka yra itin konkurencinga, optimalus kiekis gali būti žemesnis nei rinkoje, kur nėra tokio konkurencinio spaudimo. Be to, tai taip pat gali būti susiję su produkto ar paslaugos tipu, klientų charakteristikomis ir kitais veiksniais. Kadangi telekomunikacijų rinka yra labai konkurencinga, jos vidutinis Churn'erių kiekis kasmet yra maždaug tarp 20- 40%[2]. Analizuojamas turimas duomenų rinkinys:



8 pav.: Churnerių kiekis

Pasirinkto telekomunikacijų tiekėjo klientų išlaikymo rezultatai yra žymiai geresni nei vidutinis rinkos lygis, taip pat pastebima 10% Churn'erių, kuriuos reikėtų atsižvelgti vertinant duomenų rinkinio išbalansuotumą ir tikslumo prognozavimo kokybę (žr. 8 pav.). Šiuo atveju, F1-score rodikliai yra naudingi, kadangi leidžia kokybiškai įvertinti tokių problemų reikšmę.

5.3 Koreliacijos su Churn

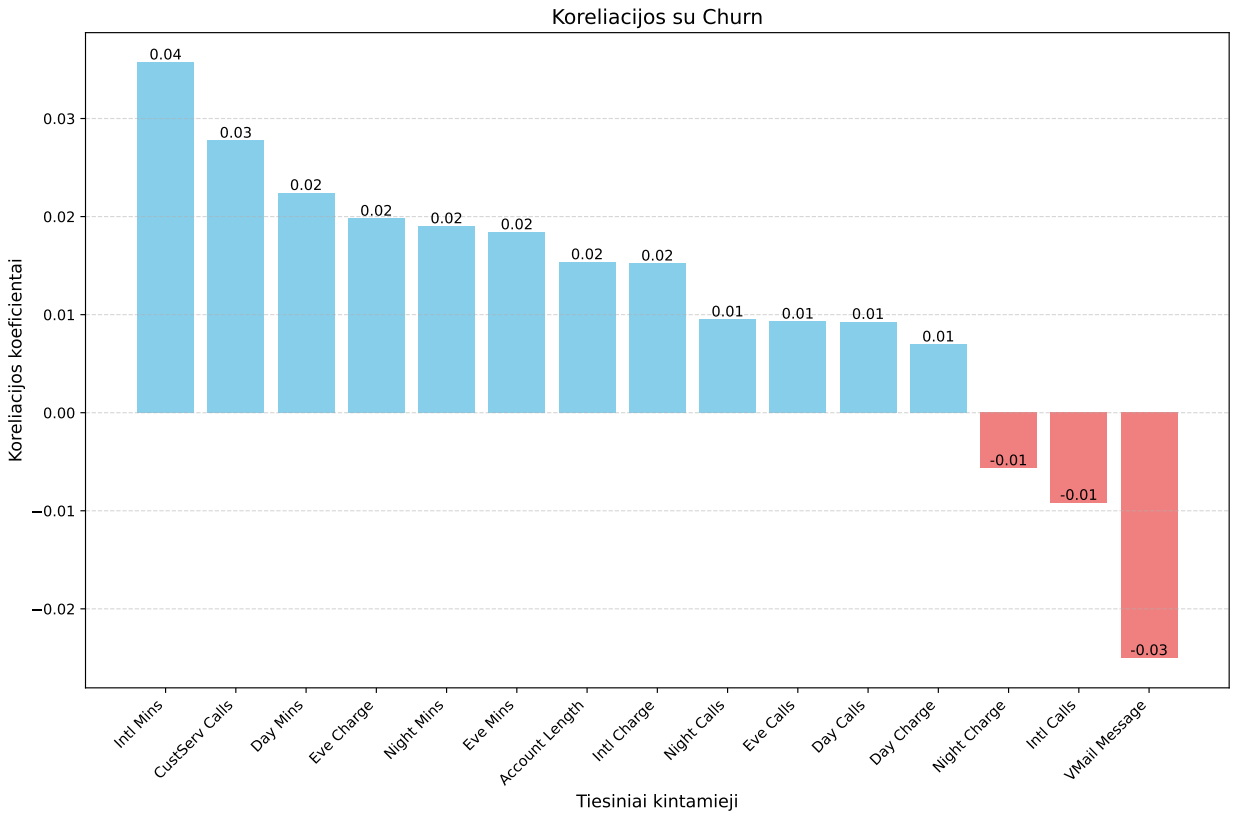
Analizuodami kitus atributus ir jų ryšius su Churn reiškiniu, naudojame koreliacijų analizę, siekdami išsiaiškinti ryšius ir tendencijas, kurios padės supaprastinti uždavinį ir nustatyti aktualiausias savybes (žr. 9 pav.).

$$r_{pb} = \frac{M_1 - M_0}{s_n} \sqrt{\frac{n_1 n_0}{n^2}} \tag{11}$$

,čia s_n yra standartinis nuokrypis

$$s_n = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (X_i - \overline{X})^2} \tag{12}$$

‚M_1 yra kintamojo vidutinė reikšmė pirmoje grupėje, o M_2 yra kintamojo vidutinė reikšmė antroje grupėje.

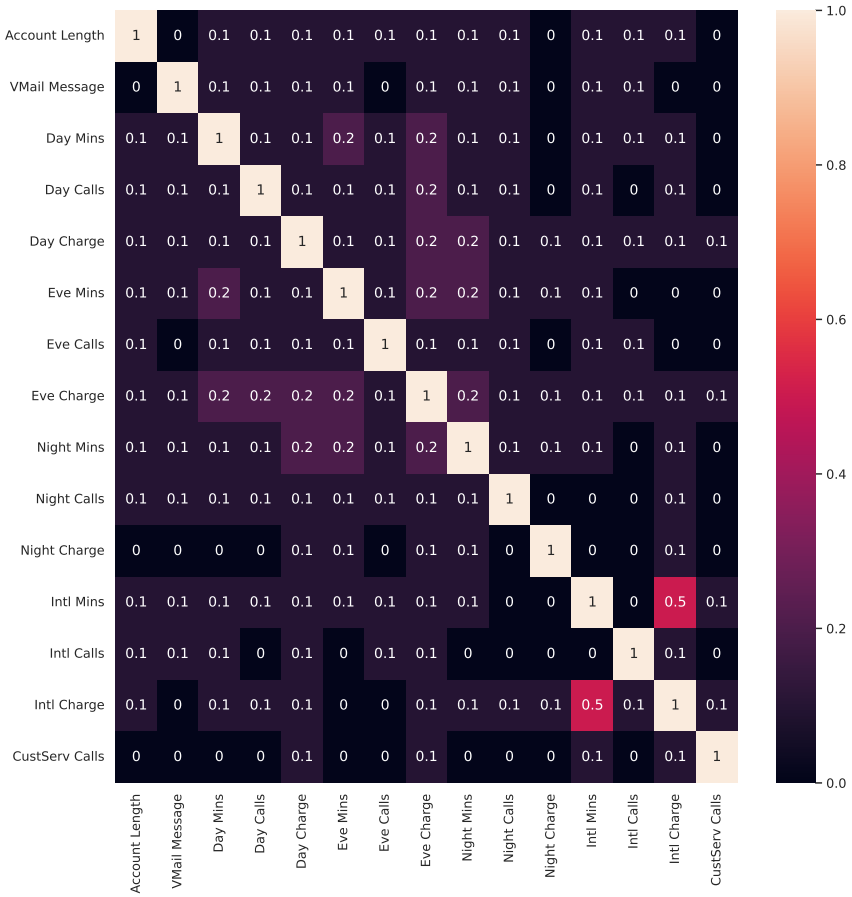


9 pav.: Koreliacijos

Išanalizavus koreliacijų grafiką, pastebėta, kad nėra stipriai koreliuojančių atributų su Churn reiškiniu. Tai rodo, kad uždavinys yra sudėtingesnis, ir reikalingi kiti metodai, leidžiantys geriau identifikuoti savybes ir/ar jų kombinacijas, kurios galėtų nustatyti pagrindines Churn priežastis.

5.3.1 Koreliacijos tarp kintamųjų

Toliau analizuojamos koreliacijos tarp visų kintamųjų ir braižomas šilumos žemėlapis (angl. *heat map*), kuris vizualiai padeda indentifikuoti ryšius tarp kintamųjų (žr. 10 pav.).



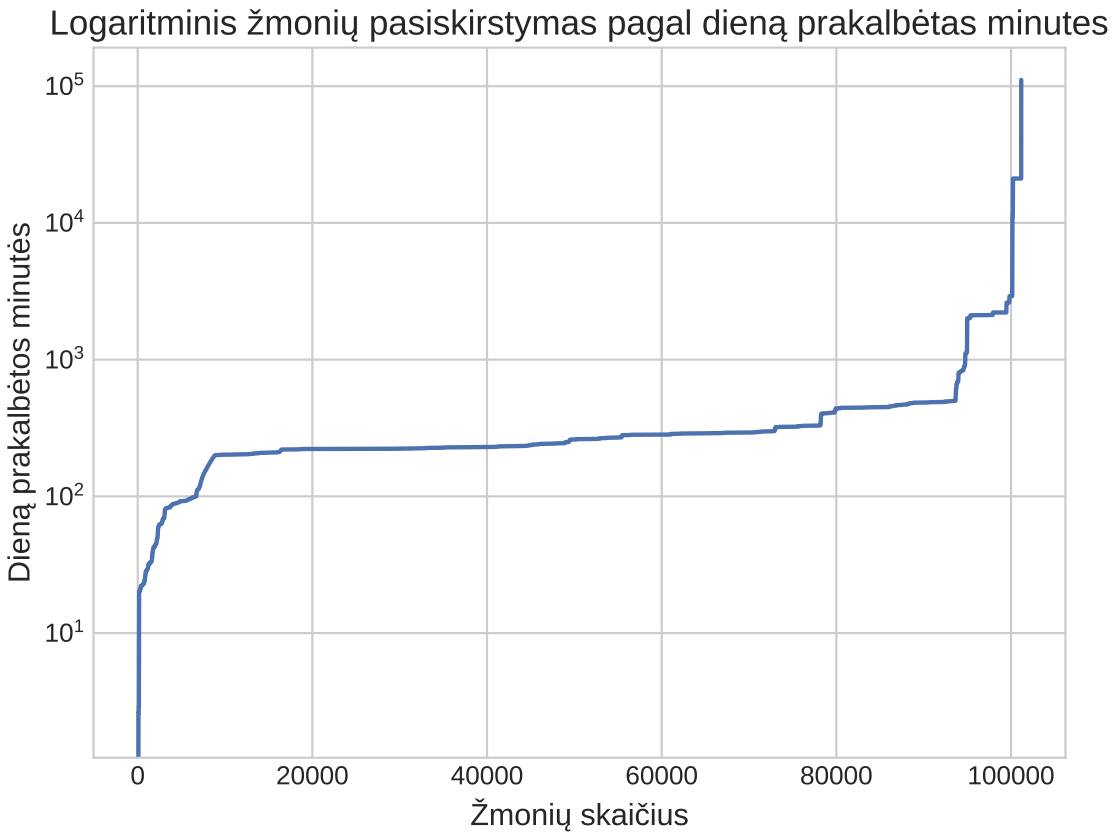
10 pav.: Koreliacijų matrica

Iš grafiko matome, kad beveik nėra koreliuojančių parametrų, kas yra ganėtinai keista, kadangi tikimės, kad didėjant skambučių kiekiui dienos/vakaro/nakties metų, turėtų augti ir prakalbėtas laikas. Galimas atvejis, kad duomenyse yra tiek vartotojai, kurie naudojasi paslaugomis tik asmeniniais tikslais, tiek verslo klientai, kurie naudojasi pagalbinėmis priemonėmis, vykdo ilgus skambučius skirtus perduoti informaciją, ir taip sukuria priešingą atvejį – mažai skambučių ir ilgas prakalbėtas laikas. Tą taip pat pagrindžia vienintelė stipresnė koreliacija tarp kintamųjų – skambučių kiekis į pagalbos centrus ir prakalbėtas laikas su jais, kadangi atkrenta verslo naudojamos programos, kuriems neaktualu skambinti į pagalbos centrus.

Iš grafiko pastebime, kad yra labai mažai koreliuojančių parametrų, kas yra gana netikėta, atsižvelgiant į tai, kad tikėtina, jog didėjant skambučių kiekiui dienos/vakaro/nakties metu, turėtų didėti ir prakalbėtas laikas. Galimas scenarijus yra toks, kad duomenyse yra tiek vartotojų, kurie naudojasi paslaugomis tik asmeniniais tikslais, tiek verslo klientų, kurie naudojasi pagalbinėmis priemonėmis. Pastarieji, vykdydami ilgus skambučius, skirtus peruoti informaciją, sukuria atvirkštinę situaciją - mažai skambučių ir ilgas prakalbėjimo laikas. Tą taip pat pagrindžia vienintelė stipresnė koreliacija tarp kintamųjų – skambučių kiekis į pagalbos centrus ir prakalbėtas laikas su jais, nes biznio naudojamos pagalbinės priemonės dažniausiai nevykdo skambučių į pagalbos centrus, kadangi tai nėra aktualu jų veiklai.

5.3.2 Duomenų pasiskirstymas

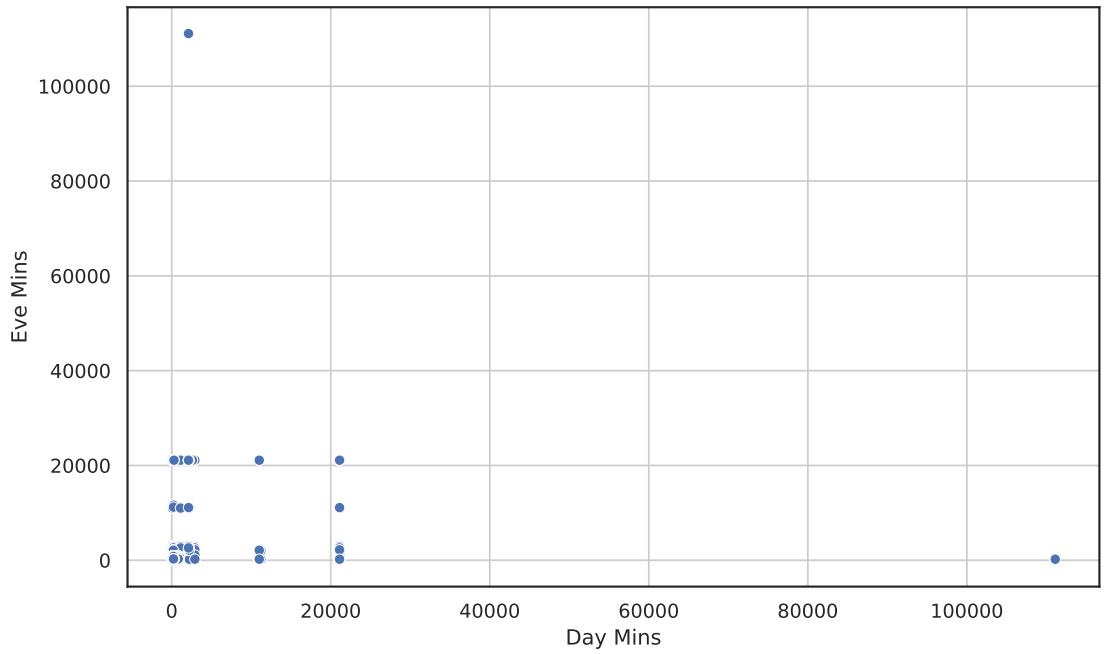
Detaliau analizuojame klientus ir jų prakalbėtą laiką dienos metu.



11 pav.: Logaritminis pasiskirstymas

Pastebime, kad didžioji dalis klientų (apie 90%) per visą laikotarpį naudojasi paslaugomis iki 1000 minučių, o likusi mažuma klientų yra prakalbėję daugiau nei 100 000 minučių. Tikėtina, kad dalis klientų yra didelės korporacijos, naudojančios paslaugas verslo tikslais.

Analizuojama sklaidos diagrama, siekiant išsiaiškinti duomenų išsibarstymą:



12 pav.: Sklaidos diagrama

Pastebime, kad duomenys yra išsibarstę mažais klasteriais, o tao parodo, kad:

1. Vartotojai yra labai akivaizdžiai susiskirstę į unikalias grupeles.
2. Duomenys yra specifiškai ištraukti iš didesnio duomenų rinkinio su apribojimais.
3. Buvo naudotas *Oversampling* metodas, generuojantis panašius vartotojus.

Tikėtina, kad buvo naudojamas *Oversampling* metodas, kuris praturtina aibę sintetiniais duomenimis, taip pagerindamas mašininio mokymosi rezultatus, jei šis metodas taikomas teisingai. Pagrindinė rizika su *Oversampling* yra ta, kad modelis gali per daug prisitaikyti prie duomenų ir atlikti klaidingas predikcijas su testavimo duomenų rinkiniu[7].

5.3.3 Išskirčių analizė

Kadangi duomenų pasiskirstymas yra specifinis, atliekama išskirčių analizė, siekiant identifikuoti dalį duomenų, kurie gali būti klasifikuojami kaip išskirtys.

Pirmuoju būdu taikoma yra Gauso Z statistika, tikrinanti ar elementas yra statistiškai reikšmingai nutolęs nuo vidurkio.

$$Z = \frac{x - \mu}{\sigma} \tag{13}$$

, kur μ yra imties vidurkis, o σ yra standartinis nuokrypis.

Kritinę reikšmę nustatoma 1,96, kuri yra α=0.05 lygmens dvipusis pasikliautinasis intervalas. Gaunami šie rezultatai:

Identifikuota 12263 vartotojai, kurie nepatenka į pasikliautinąjį intervalą, o tai sudaro apie 12% viso duomenų rinkinio.

Antrasis būdas – kvantilių metodas, kuris apskaičiuoja 25 ir 75 lygio kvartilius ir pagal juos identifikuoja išskirtis. Matematine formule išreikštas išskirčių apskaičiavimas, naudojant kvantilių metodą:

$$Pasikliautinasis \quad intervalas = [Q_{25} - coef * (Q_{75} - Q_{25}) \quad ; \quad Q_{75} + coef * (Q_{75} - Q_{25})] \tag{14}$$

Čia $coef$ yra parenkamas koeficientas, nustatantis kaip griežtai/atlaidžiai metodas identifikuos, kas yra išskirtis, dažniausiu atveju reikšmė yra parenkama 1,5. Didinat šį koeficientą, funkcija tampa mažiau jautri išskirtims, nes yra plečiamas priimtinų reikšmių intervalas. Kai koeficientas mažinamas – atvirkščiai.

Kai $coef = 1{,}5$, gaunami rezultatai, kad identifikuota 60306 išskirčių, kurios sudaro didžiąją dalį turimų duomenų.

Siekiant neprarasti daugelio duomenų nuspęsta išskirčių nešalinti duomenų, nes be papildomos analizės nėra aišku, kokiu būdu yra gauti duomenys.

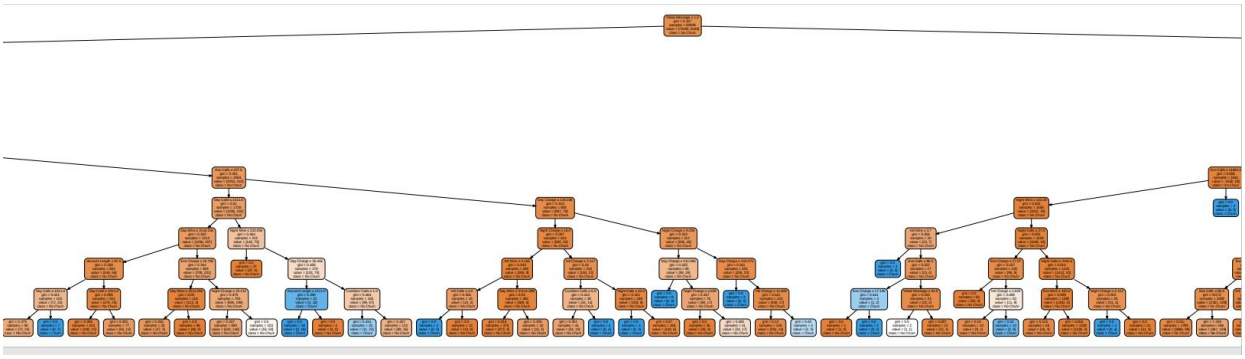
5.4 Mašininio mokymosi rezultatai

5.4.1 Sprendimų medžiai

Praktinėje tyrimo dalyje siekiama įvertinti sprendimų medžių efektyvumą prognozuojant klientų Churn'ą, naudojant realaus pasaulio duomenų rinkinį. Sprendimų medžiai - tai mašininio mokymosi algoritmo tipas, kuris gali apdoroti ir kategorinius, ir skaitinius duomenis, atlikti automatinę požymių atranką ir yra lengvai interpretuojamas bei vizualizuojamas. Duomenų aibę suskirstysime į mokymo ir testavimo aibes, mokymo aibėje išmokysime sprendimų medžio klasifikatorių ir įvertinsime jo veikimą testavimo aibėje naudodami keletą rodiklių. Taip pat eksperimentuosime su sprendimų medžio gylio apribojimu, kad išvengtume persimokymo, ir palyginsime sprendimų medžio algoritmo našumą su kitais mašininio mokymosi modeliais. Šio tyrimo rezultatai suteiks įžvalgų apie sprendimų medžių gebėjimą prognozuoti Churn'ą ir galimą jų naudojimą telekomunikacijos sektoriuje.

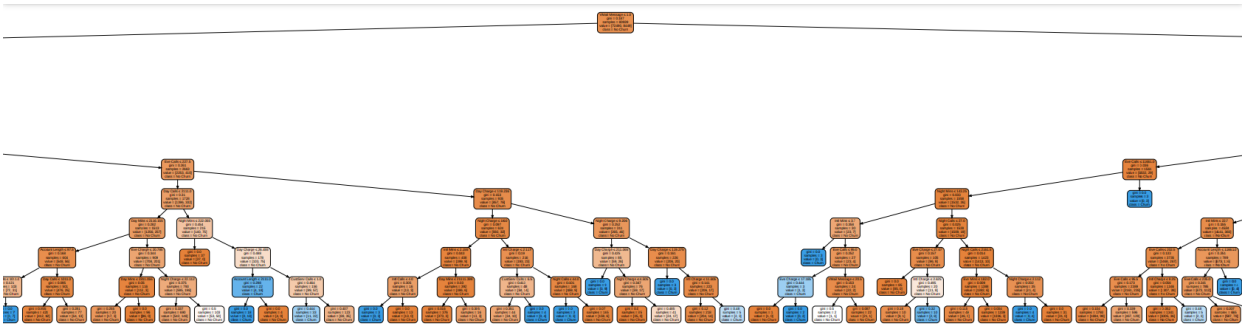
Tyrimą pradedame neribojant mokymosi gylio, t.y., leidžiame modeliui daryti duomenų padalijimus tol, kol visi klientai atsiduria tyruose lapuose, t.y. tik su savo klasės klientais, arba pasiekiamas maksimalus galimas klientų paskirstymas.

Mūsų eksperimentų rezultatai parodė, kad sprendimų medžio algoritmo pasiektas tikslumas buvo 96,7%, F1-balas – 84,1%, o jautrumas – 84,4%. Tai yra labai geras rezultatas, kuris dažniausiai nėra pasiekiamas dirbant su realiais duomenimis. Tačiau neribojant gylio medis labai stipriai išauga ir sunku įvertinti svarbiausias savybes, kurios nulemia medžio padalijimo kriterijus. Šiuo atveju gaunami 7239 lapai, kurie identifikuoja medžio sprendimą. Tai yra daug, todėl vizualizuoti tampa nepraktiška (žr. 13 pav.).



13 pav.: Sprendimo medžio dalis

Siekiant sumažinti sprendimų medį, gylį apribojame iki 10. Šiuo atveju rezultatai: tikslumas – 91,3%, F1-balas – 31,4%, jautrumas – 19,4%. Pastebime, kad rezultatai gan nukentėjo, aptinkame maždaug 2 iš 10 Churn'erių, tačiau bendras tikslumas išlieka gan aukštas. Šiuo metodu gauname 749 lapus, kas yra apie 10 kartų mažesnis lapų skaičius. Parinkti gylio apribojimai padaro modelį paparstesnį, tačiau jis vis tiek išlieka per didelis vizualizuoti (žr. 14 pav.):



14 pav.: Sprendimo medžio dalis

Kadangi modelis praranda savo tikslumą, daugiau jo mažinti neverta. Siekiama iš didesnio modelio ištraukti reikšmingiausias savybes.

5.4.2 Atsitiktinis miškas

Atsitiktinis miškas yra populiarus mašininio mokymosi metodas, kuris sujungia kelis sprendimų medžius, kad pagerintų prognozavimo rezultatus ir sumažintų persimokymą. Eksperimentinėje šio bakalauro darbo dalyje tiriamas atsitiktinio miško efektyvumas prognozuojant klasifikavimo uždavinio rezultatą. Nagrinėsime skirtingų hiperparametrų, tokių kaip medžių skaičius, medžių gylis ir požymių padalijimo metodus.

Duomenų apmokymui bei testavimui naudojame skirtingus hiperparametrus ir galimas jų kombinacijas:

1. Medžių skaičius: [50, 100, 200]
2. Maksimalus galimas gylis: [5, 10, 20, None(neribojamas)]
3. Minimalus lapo padalijimas: [2, 5, 10]
4. Minimalus elementų lape skaičius: [1, 2, 4]

Tai yra 108 galimos kintamųjų kombinacijos, iš kurių kiekvieną kombinaciją tikriname 5 kartus ir rezultatus vidurkiname, kad išvengtume atsitiktinai gero rezultato. Iš viso bus įvykdyta 540 apmokymų.

Geriausias gaunamas modelis yra su hiperparametrais:

1. Medžių skaičius – 100
2. Maksimalus galimas gylis – neribojamas
3. Minimalus lapo padalijimas – 2
4. Minimalus elementų lape skaičius – 1

Šio modelio *cross-validation* tikslumas – 98,3%. Gaunami testavimo imties modelio rezultatai: tikslumas – 95,8%, F1-balas – 81,1%, jautrumas – 87,4%. Rezultatai yra labai panašūs kaip ir sprendimų medžių atveju, kur yra neribojamas medžio gylis. Tikslumas bei F1-balas truputį nukenčia, tačiau gaunamas geresnis jautrumas, tai reiškia, kad modelis yra labiau prisitaikęs atrasti Churn'erius, išlaikydamas kitas statistikas sąlyginai gerai. Kadangi atsitiktiniai miškai yra atsparesni persimokymui bei pradinei parenkamai apmokymo imčiai bei generuoja tokius pat gerus rezultatus kaip sprendimų medžiai, galime teigti, kad atsitiktiniai miškai bus tinkamesnis modelis šiam uždaviniui spręsti, nei sprendimų medis.

5.4.3 LSTM

Modeliuojame LSTM algoritmą, kuris yra vienas iš populiariausių metodų ir generuojantis geras predikcijas. Remiamės [6] šaltiniu, kuriame nagrinėjami 8 skirtingi metodai ir lyginami metodų rezultatai, ranks nurodo kelintą vietą užėmė algoritmas. Gaunamos reikšmės:

Table 6
Average ranks of the models measured by the five performance measures.

	AUC	F1-Score	Log Loss	Lift	EMPC	Overall Average Rank
LSTM-based Model	1.00	1.00	1.00	1.00	1.00	1.00
RFM-based Model	2.00	2.03	2.00	2.00	2.23	2.05
CNN-based Model	3.00	2.97	3.00	3.07	2.77	2.96
RF-Daily	4.00	5.07	4.33	5.63	4.00	4.61
Statistics-based Model	6.00	4.00	5.90	3.97	6.00	5.17
CNN_prev	5.00	6.87	4.77	6.30	5.00	5.59
RF-Monthly	7.03	6.23	7.00	6.37	7.03	6.73
LSTM-Monthly	8.00	7.83	8.00	7.67	7.97	7.89

15 pav.: Modelių palyginimas

Visų paveikslė pateiktų kriterijų LSTM paremtas modelis generavo tiksliausius rezultatus, todėl siekiama metodą atkartoti su turimu duomenų rinkiniu.

Sukūrus sąlyginai paprastą LSTM modelį, kuriame yra 2 paslėpti sluoksniai su, atitinkamai, 200 ir 100 neuronų:

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 lstm (LSTM)                  (None, 1, 200)            172800

 lstm_1 (LSTM)                (None, 100)               120400

 dense (Dense)                (None, 1)                 101

=================================================================
Total params: 293,301
Trainable params: 293,301
Non-trainable params: 0
```

16 pav.: LSTM modelio parametrai

Turime modelį su 293301 parametrų, kurį apmokome su turimu duomenų rinkiniu. Apmokymui taikome 80% visos duomenų imties, o validacijos dalį paliekame likusius 20%. Paklaidoms skaičiuoti naudojame MSE (Mean Square Error) kriterijų. Taip pat, kad duomenys į atskiras imtis būtų traukiami atsitiktinai, visas duomenų eilutes sumaišome. Tokiu būdu išvengiame specifinių grupių priskyrimo į apmokymo bei validacijos imtis.

```
Epoch 1/10
1265/1265 [==============================] - 38s 26ms/step - loss: 0.0937 - accuracy: 0.8956 - val_loss: 0.0923 - val_accuracy: 0.8971
Epoch 2/10
1265/1265 [==============================] - 19s 15ms/step - loss: 0.0936 - accuracy: 0.8956 - val_loss: 0.0924 - val_accuracy: 0.8971
Epoch 3/10
1265/1265 [==============================] - 20s 16ms/step - loss: 0.0936 - accuracy: 0.8956 - val_loss: 0.0924 - val_accuracy: 0.8971
Epoch 4/10
1265/1265 [==============================] - 19s 15ms/step - loss: 0.0935 - accuracy: 0.8956 - val_loss: 0.0924 - val_accuracy: 0.8971
Epoch 5/10
1265/1265 [==============================] - 21s 16ms/step - loss: 0.0935 - accuracy: 0.8956 - val_loss: 0.0924 - val_accuracy: 0.8971
Epoch 6/10
1265/1265 [==============================] - 19s 15ms/step - loss: 0.0935 - accuracy: 0.8956 - val_loss: 0.0924 - val_accuracy: 0.8971
Epoch 7/10
1265/1265 [==============================] - 20s 16ms/step - loss: 0.0935 - accuracy: 0.8956 - val_loss: 0.0924 - val_accuracy: 0.8971
Epoch 8/10
1265/1265 [==============================] - 19s 15ms/step - loss: 0.0935 - accuracy: 0.8956 - val_loss: 0.0925 - val_accuracy: 0.8971
Epoch 9/10
1265/1265 [==============================] - 20s 16ms/step - loss: 0.0935 - accuracy: 0.8956 - val_loss: 0.0925 - val_accuracy: 0.8971
Epoch 10/10
1265/1265 [==============================] - 19s 15ms/step - loss: 0.0935 - accuracy: 0.8956 - val_loss: 0.0924 - val_accuracy: 0.8971
2530/2530 [==============================] - 10s 4ms/step
633/633 [==============================] - 2s 3ms/step
Training accuracy: 89.56%
Training F1-score: 0.00%
Training Recall: 0.00%
Validation accuracy: 89.71%
Validation F1-score: 0.00%
Validation Recall: 0.00%
```

17 pav.: LSTM modelio rezultatai

Pastebime, kad apmokant modelį beveik nekinta nei tikslumas, nei paklaidos funkcija. Modelis neįdentifikuoja nei vieno Churn'erio, kadangi F1-balas ir jautrumo reikšmės yra 0. Tai reiškia, kad reikia modifikuoti modelį, siekiant identifikuoti Churn'erius.

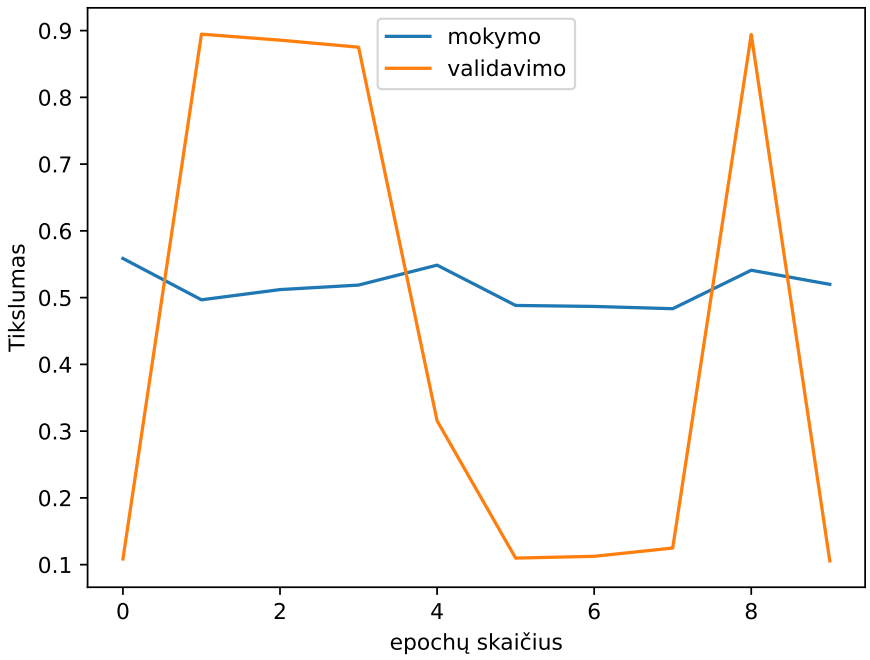
Modifikuojame metodą pridedant Churn'eriams svorius, kad jie būtų reikšmingesni modeliui, kadangi tikslas identifikuoti kuo didesnę dalį žmonių, kurie gali atsisakyti paslaugų.

Kadangi Churn'erių yra 8,5 karto mažiau, atitinkamai pridedamas toks svoris, kad modelis būtų labiau subalansuotas.

```
Epoch 1/10
1265/1265 [==============================] - 39s 26ms/step - loss: 0.4575 - accuracy: 0.5587 - val_loss: 0.2773 - val_accuracy: 0.1085
Epoch 2/10
1265/1265 [==============================] - 31s 24ms/step - loss: 0.4501 - accuracy: 0.4967 - val_loss: 0.2009 - val_accuracy: 0.8946
Epoch 3/10
1265/1265 [==============================] - 21s 16ms/step - loss: 0.4498 - accuracy: 0.5120 - val_loss: 0.2295 - val_accuracy: 0.8857
Epoch 4/10
1265/1265 [==============================] - 22s 18ms/step - loss: 0.4493 - accuracy: 0.5188 - val_loss: 0.2461 - val_accuracy: 0.8752
Epoch 5/10
1265/1265 [==============================] - 21s 16ms/step - loss: 0.4487 - accuracy: 0.5487 - val_loss: 0.2528 - val_accuracy: 0.3156
Epoch 6/10
1265/1265 [==============================] - 23s 18ms/step - loss: 0.4489 - accuracy: 0.4883 - val_loss: 0.2644 - val_accuracy: 0.1098
Epoch 7/10
1265/1265 [==============================] - 22s 17ms/step - loss: 0.4488 - accuracy: 0.4868 - val_loss: 0.2689 - val_accuracy: 0.1125
Epoch 8/10
1265/1265 [==============================] - 19s 15ms/step - loss: 0.4486 - accuracy: 0.4834 - val_loss: 0.2556 - val_accuracy: 0.1249
Epoch 9/10
1265/1265 [==============================] - 21s 17ms/step - loss: 0.4489 - accuracy: 0.5410 - val_loss: 0.2327 - val_accuracy: 0.8941
Epoch 10/10
1265/1265 [==============================] - 20s 16ms/step - loss: 0.4490 - accuracy: 0.5199 - val_loss: 0.2661 - val_accuracy: 0.1056
2530/2530 [==============================] - 12s 4ms/step
633/633 [==============================] - 3s 4ms/step
Training accuracy: 10.39%
Training F1-score: 18.81%
Training Recall: 100.00%
Validation accuracy: 10.56%
Validation F1-score: 19.07%
Validation Recall: 100.00%
```

18 pav.: Svorinio LSTM modelio rezultatai

Modelis identifikavo visus Churn'erius, kadangi modelio jautrumo parametras yra 100%. Tačiau kiti rodikliai yra prasti. Apmokymo ir validacijos imtis vizualizuojame, kad pastebėtume tendencijas:



19 pav.: Svorinio LSTM tikslumas

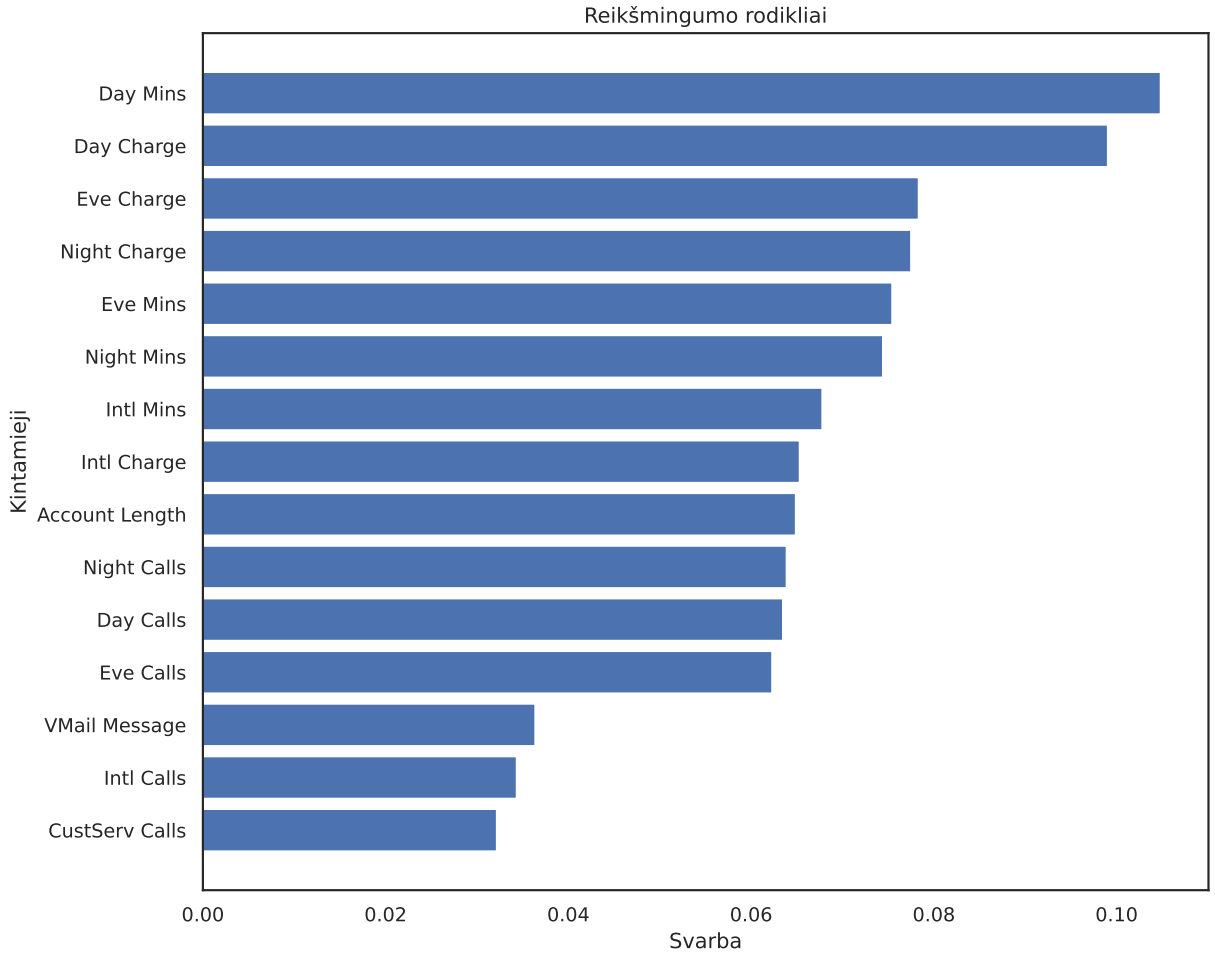
Iš validacijos imties pastebime, kad modelis generuoja nuo (10% iki 90%) tikslumą, priklausomai nuo epochos, kurioje būtų nutrauktas mokymasis. Atidžiau pažvelgus, tikslumo reikšmės 10% ir 90% yra neatsitiktinės, kadangi modeliui priskyrėme vienodus svorius. Dėl to jis nesugeba nuspręsti, kuris klientas yra svarbesnis, neranda raktinių charakteristikų, pagal ką galėtų nustatyti, ar klientas Churn'ins, ar ne, todėl jis tiesiog kaip predikciją teigia, kad arba visi pasiliks, arba visi Churn'ins. Toks modelis yra nepastovus ir nepatikimas, siekiant tiksliai nuspręsti, kurie žmonės yra linkę Churn'inti.

Galime teigti, kad LSTM metodas nėra pritaikytas šio formato duomenims nuspėti, kadangi jis yra skirtas duomenų analizei laike (kaip kinta kiekviena savybė laike), o kadangi mūsų duomenys yra agreguoti kliento lygmenyje, ši savybė yra prarandama. Tačiau esant tokiems duomenims, žinoma, kad metodas leidžia gauti gerus rezultatus[6].

5.5 Atsitiktinio miško reikšmingumo rodikliai

Reikšmingumo rodikliai bus apskaičiuojami naudojant Gini svarbos prieaugį, tai paprasčiausias būdas interpretuoti turimus rodiklius, kadangi jų suma susideda į 1, t.y., normuoti, todėl lengvai galima lyginti tarpusavyje ir išreikšti procentine dalimi.

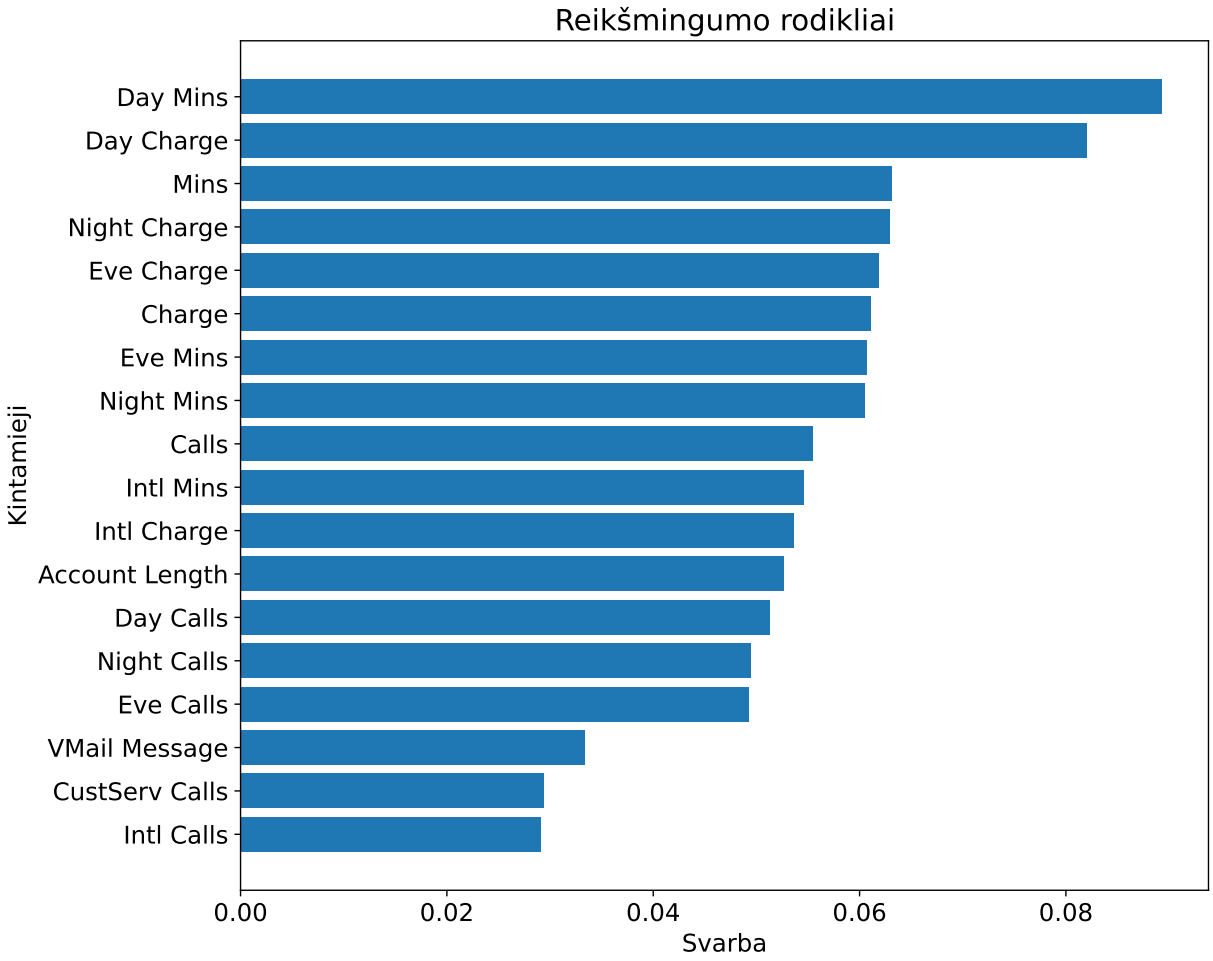
Pradinė analizė atliekama su visais kintamaisiais ir parinktu geriausiu atsitiktinių miškų algoritmu (žr. skyrių 5.4.2 Atsitiktinis miškas). Gaunami rezultatai:



20 pav.: Reikšmingumo rodikliai

Pastebime, kad svarbiausi kriterijai, geriausiai atskiriantys Churn'erius iš populiacijos yra per dieną prakalbėtų minučių skaičius bei dienos metu sumokėta suma. Visi kiti kintamieji turi maždaug tokį patį svorį – yra vienodai svarbūs prognozėms. Tik kintamieji – skambučiai į užsienį, skambučiai į pagalbos centrus ir balso žinutės nėra tokie reikšmingi.

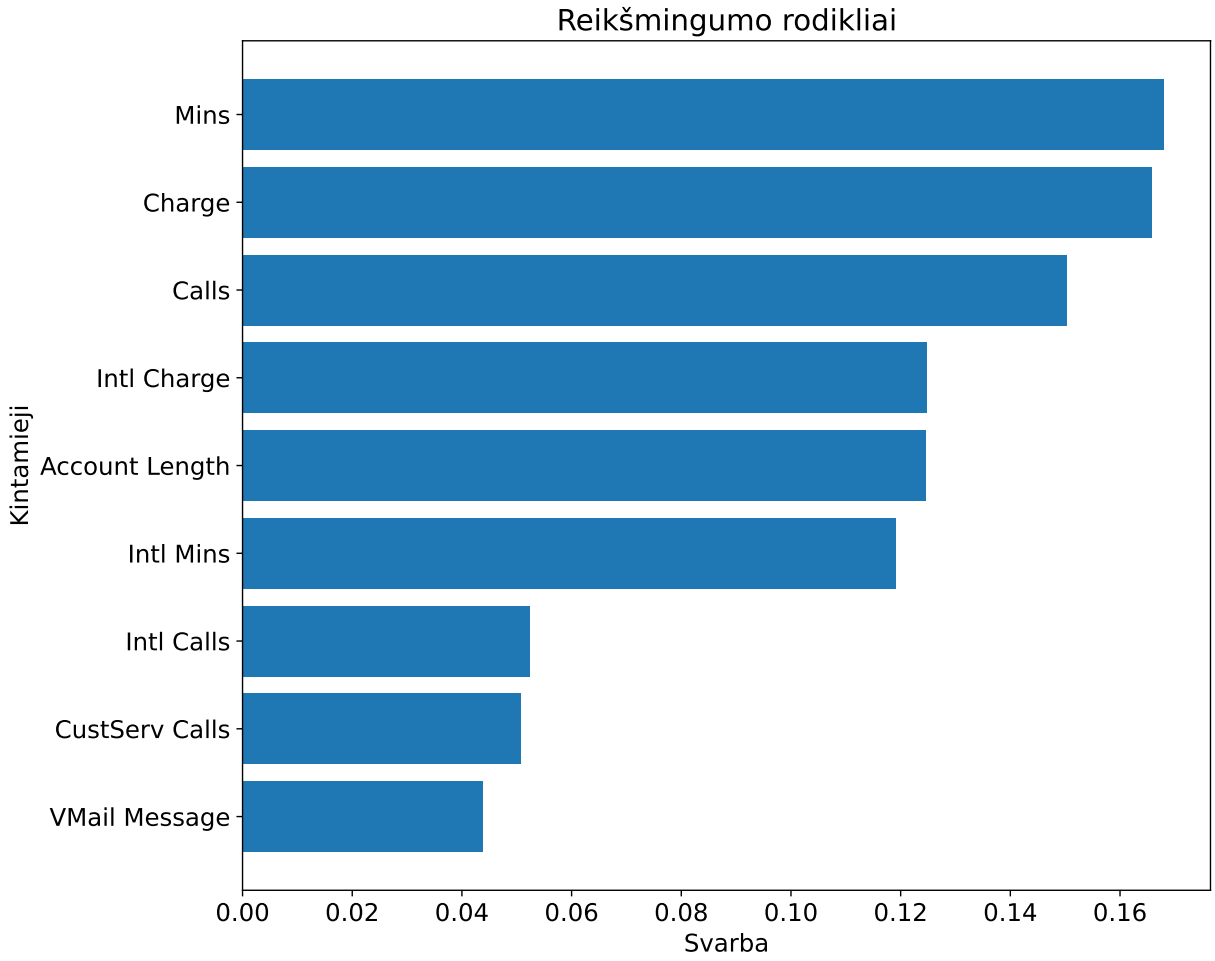
Modifikuojame duomenų rinkinį agreguodami dienos, vakaro ir nakties skambučius, prakalbėtą laiką ir sumokėtą sumą į 3 papildomus dydžius – *Calls, Mins, Charge*. Pakartotinai, naudojant tuos pačius parametrus sukuriame atsitiktinių miškų algoritmą ir apskaičiuojame reikšmingumus. Naujai gauti rezultatai:



21 pav.: Reikšmingumo rodikliai su papildomais kintamaisiais

Atlikus analizę, pastebėta, kad rezultatai beveik nepakito, predikcijų tikslumas išlieka toks pats, o reikšmingumo kriterijų tendencija nekinta, išlieka tie patys reikšmingi, bei mažiau reikšmingi kintamieji.

Tada pašaliname agregavimui skirtus kintamuosius, tačiau paliekant naujus agreguotus, kad būtų galime identifikuoti, kas yra svarbiau, prakalbėtos minutės, sumokėta suma ar skambučių skaičius. Gaunami rezultatai:



22 pav.: Agreguoti reikšmingumo rodikliai

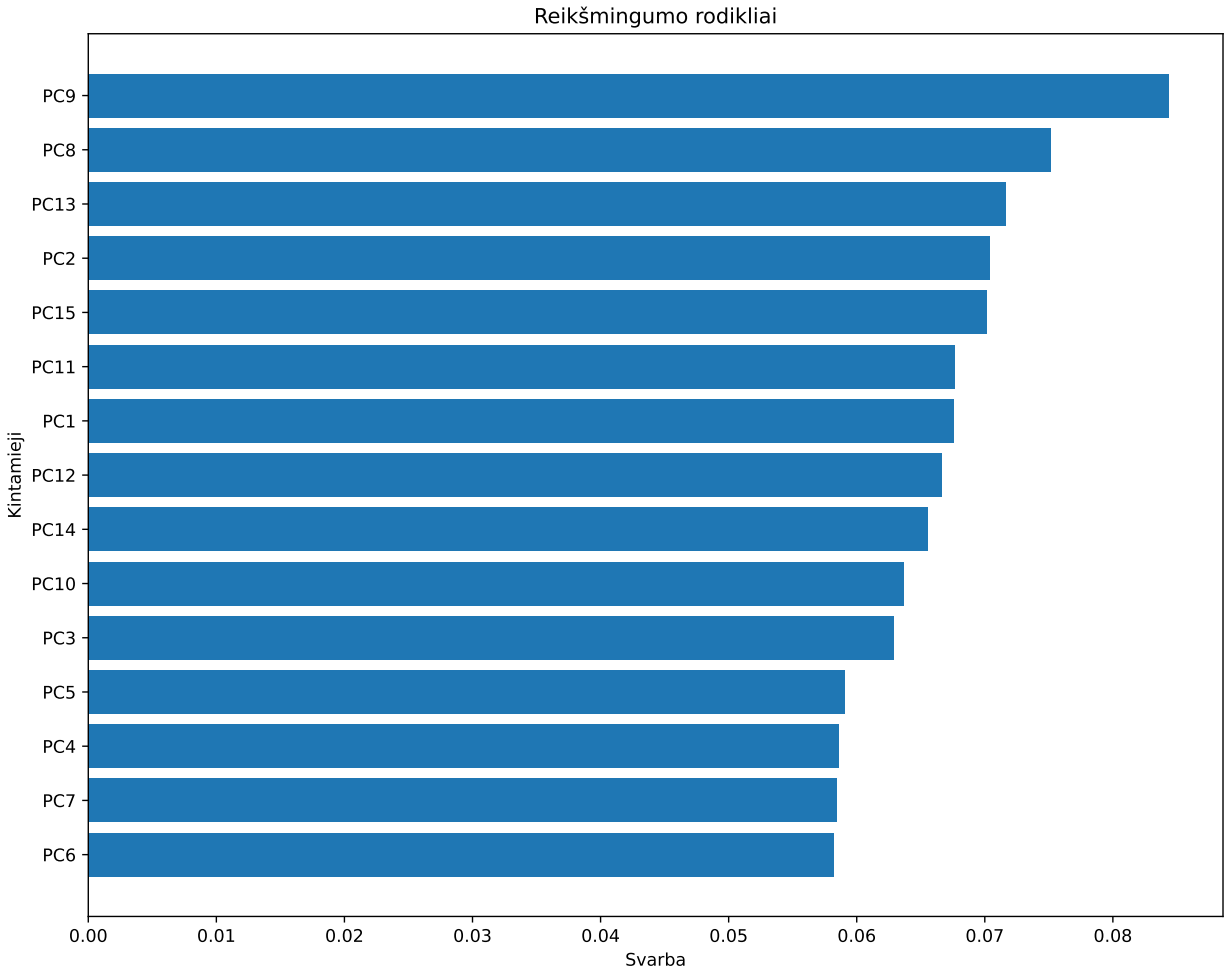
Šiuo atveju pastebime, kad agreguoti kintamieji sudaro svarbiausią dalį visų kintamųjų. Predikcijos rodikliai beveik nekinta, todėl daug informacijos neprarandame, tačiau modelis tampa truputį paprastesnis suvokimui. Tendencijos išlieka tokios pačios, kad pagal bendrą prakalbėtą minučių skaičių galime identifikuoti daugiausiai Churn'erių.

5.6 Pagrindinių komponenčių analizė

Pagrindinių komponenčių analizė (PCA) yra dimensijų mažinimo metodas, dažnai naudojamas mašininio mokymosi srityje, taip pat ir kartu su atsitiktinio miško algoritmu. PCA gali pagerinti atsitiktinių miškų rezultatus sumažindama įvesties duomenų dimensiją ir kartu išsaugodama svarbius modelius ir ryšius. Pradinius požymius paverčiant nauju nekoreliuotų kintamųjų, vadinamų pagrindiniais komponentais, rinkiniu, PCA gali užfiksuoti informatyviausius duomenų

aspektus ir atmesti nereikalingus ar triukšmingus požymius. Dėl to gali pagerėti modelio veikimas, nes sumažėja perteklinis pritaikymas, pagerėja aiškinimas ir padidėja skaičiavimo efektyvumas. Be to, PCA gali padėti spręsti multikolinearumo problemas, nes sprendžiami didelės kintamųjų koreliacijos klausimai. Įtraukus PCA į atsitiktinio miško algoritmo išankstinio apdorojimo procesą, galima efektyviau ir veiksmingiau modeliuoti, sutelkiant dėmesį į svarbiausius duomenų aspektus.

Kadangi PCA gali pagerinti predikcijos rezultatus, pradžioje analizę atliekame su maksimaliu komponenčių skaičiumi – 15, kad nebūtų prarandami duomenys ir gaunami geriausi rezultatai (žr. 23 pav.).



23 pav.: PCA reikšmingumo rodikliai

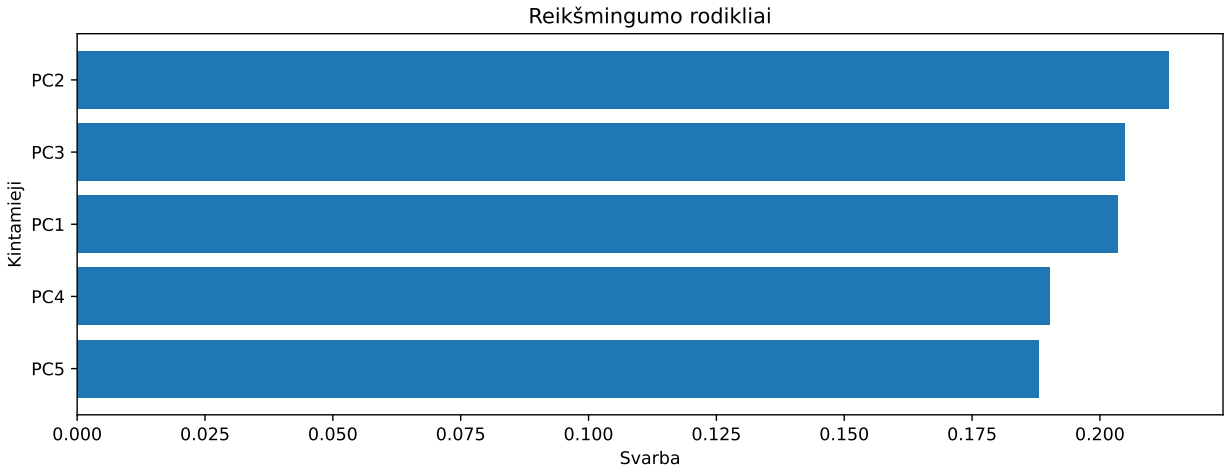
Pastebime, kad visos pagrindinės komponentes yra panašaus svarbumo, tik *PC*9 yra šiek tiek reikšmingesnė. Ištyrus jos tiesinę kombinaciją su originaliais kintamaisiais pastebime, kad jos pagrindinę dalį sudaro *Vmail Message* – 0.998, o visi likę nereikšmingą dalį. Kas keisčiausia, kad

ši komponentė tampa pati svarbiausia, norint identifikuoti Churn'erius, o kitais metodais pastebėta, kad ji yra mažiausiai reikšminga. Todėl galime teigti, kad originalių duomenų svarbumas priklauso nuo parinkto apskaičiavimo metodo ir kad visi kintamieji yra svarbūs, siekiant identifikuoti Churn'erius. Šiuo metodu gauti predikcijų efektyvumo rodikliai:

1. Tikslumas – 95,8%
2. F1-balas – 81,1%
3. Jautrumas – 87,2%

Tai yra minimaliai prastesnis rezultatas, nei naudojant tik pradinius duomenis.

Siekiant atrasti reikšmingus kintamuosius, sumažiname pagrindinių komponenčių skaičių iki 5 (žr. 24 pav.), taip pagreitės skaičiavimai ir galimai pavyks atskirti reikšmingiausias komponentes.



24 pav.: PCA reikšmingumo rodikliai

Pastebime, kad visos pagrindinės komponentės yra panašiai reikšmingos, gaunami efektyvumo rodikliai:

1. Tikslumas – 95,6%
2. F1-balas – 80,3%
3. Jautrumas – 86%

Rezultatai išlieka geri ir pavyksta gan stipriai apkarpyti duomenis, sumažinant duomenų kiekį 3 kartus. Skaičiavimai pagreitėja apie 20% (nuo 3min ir 30s iki 2min ir 54s).

Su PCA pagerinti predikcijų nepavyko, tačiau išsiaiškinta, kad taikant skirtingus metodus svarbiausi kintamieji gali skirtis, todėl visi kintamieji, kurie padeda identifikuoti Churn'erius, yra aktualūs.

5.7 Persimokymo analizė

Persimokymo (angl. *Overfitting*) analizė yra labai svarbus žingsnis vertinant mašininio mokymosi modelių veikimą. Analizės metu įvertinama, kiek modelis iš mokymo duomenų išmoko konkrečių dėsningumų, iki kol prarado gebėjimą apibendrinti. Persimokymo scenarijaus atveju modelis tampa pernelyg sudėtingas, fiksuoja mokymo duomenų triukšmą ir ypatumus, kurių gali nebūti nepastebėtuose duomenyse. Dėl to modelis prastai veikia, kai taikomas naujiems duomenims. Norint nustatyti ir sumažinti persimokymo poveikį, taikomi įvairūs metodai, pavyzdžiui, modelio veikimas vertinamas atskiruose validacijos arba testavimo duomenų rinkiniuose, atliekama kryžminė validacija (angl. *cross-validation*). Persimokymo analizės tikslas – rasti pusiausvyrą tarp modelio sudėtingumo ir apibendrinimo, užtikrinant, kad modelis galėtų veiksmingai apibendrinti mokymo duomenų modelius nematytiems duomenims.

5.7.1 Atsitiktinių miškų persimokymas

Kadangi gauti rezultatai geriausi su atsitiktinių miškų algoritmu, jam bus atliekama persimokymo analizė.

Atsitiktiniai miškai naudoja keletą sustabdymo kriterijų, pagal kuriuos nustatoma, kada nutraukti atskirų sprendimų medžių augimą:

1. Didžiausias medžio gylis: kiekvienas atsitiktinio miško sprendimų medis auginamas tol, kol pasiekiamas nustatytas didžiausias gylis. Tai apsaugo nuo persimokymo, nes apribojamas atskirų medžių sudėtingumas.

2. Mažiausias imčių skaičius padalijimui: sprendimų medžio mazgas toliau skaidomas tik tada, jei jame yra minimalus imčių skaičius. Šiuo kriterijumi užtikrinama, kad padalijimas būtų prasmingas, ir sumažinama persimokymo tikimybė.

3. Mažiausias pavyzdžių skaičius lape: kai mazgas padalijamas, tikrinama, ar kiekviename iš jo mazgų yra minimalus imčių skaičius. Jei mazgo dukterinis mazgas nesiekia šios ribos, tolesnis skaidymas sustabdomas ir mazgas tampa lapu.

4. Didžiausias lapinių mazgų skaičius: kai lapinių mazgų skaičius pasiekia šią ribą, medžio augimas sustabdomas.

5. Mažiausias priemaišų pagerėjimas: skirstymo kokybei įvertinti naudojamas priemaišų matas, pavyzdžiui, Gini priemaiša arba entropija. Jei potencialaus skaidymo metu pasiektas priemaišų pagerėjimas (pvz., Gini priemaišų sumažėjimas) nesiekia tam tikros ribos, skaidymo procesas sustabdomas.

6. Didžiausias medžių skaičius: taip pat galima nustatyti fiksuotą atsitiktinio miško medžių skaičių. Kai šis skaičius pasiekiamas, algoritmas nustoja auginti naujus medžius.

Šie sustabdymo kriterijai padeda kontroliuoti atsitiktinio miško dydį ir sudėtingumą bei užkirsti kelią persimokymui, užtikrinant, kad modelis būtų gerai apibendrintas nematytiems duomenims. Konkrečius kriterijus ir jų reikšmes galima koreguoti atsižvelgiant į duomenų rinkinį ir gaunamus rezultatus.

Pagal anksčiu testuotus hiperparametrus (žr. 5.4.2 skyrių), pastebime, kad geriausias modelis parenkamas beveik maksimaliai overfittint'as. Todėl pasidaro aktualu spręsti overfittinim'o problemą, kadangi medžio gylis neribojamas, medžių skaičius ganėtinai didelis, minimalus padalijimas parenkamas 2 ir minimalus elementų skaičius lape taip pat yra 1.

5.7.2 Apkarpymas

Apkarpymas (angl. *Pruning*) – tai mašininio mokymosi metodas, naudojamas siekiant išspręsti persimokymo problemą, kuri atsiranda, kai modelis tampa pernelyg sudėtingas ir mokymo duomenyse užfiksuoja triukšmą ar nereikšmingus modelius. Apkarpymas apima selektyvų tam tikrų sprendimų medžio dalių pašalinimą arba kitų tipų modelių sudėtingumo sumažinimą. Šis procesas padeda išvengti persimokymo skatindamas apibendrinimą ir mažindamas modelio jautrumą mokymo duomenims. Apkarpymo metodais siekiama rasti tinkamą pusiausvyrą tarp modelio sudėtingumo ir našumo naudojant nematytus duomenis, todėl padidėja tikslumas ir pagerėja apibendrinimo galimybės. Veiksmingai mažindamas persimokymą, atpjovimas padidina modelio gebėjimą tiksliau prognozuoti naujus, nematytus atvejus[9].

Kadangi geriausias parinktas modelis yra sudėtingas ir turėtų būti permokintas, testuojame atvirkščiai naudojant *pruning* metodą – mažinant apribojimus, siekiant patikrinti ar modelį įmanoma permokinti ir kaip kinta rezultatai apsunkinant modelį.

Kadangi visi kiti hiperparametrai jau yra neribojami arba yra minimalūs, tai koreguojamas tik medžių skaičius:

1 lentelė: Atsitiktinio miško medžių skaičius

Random Forest			
Medžių skaičius	100	200	1000
Accuracy	95,8%	95,8%	95,8%
Recall	87,4%	87,5%	87,5%
F1-score	81,1%	81,1%	81,2%

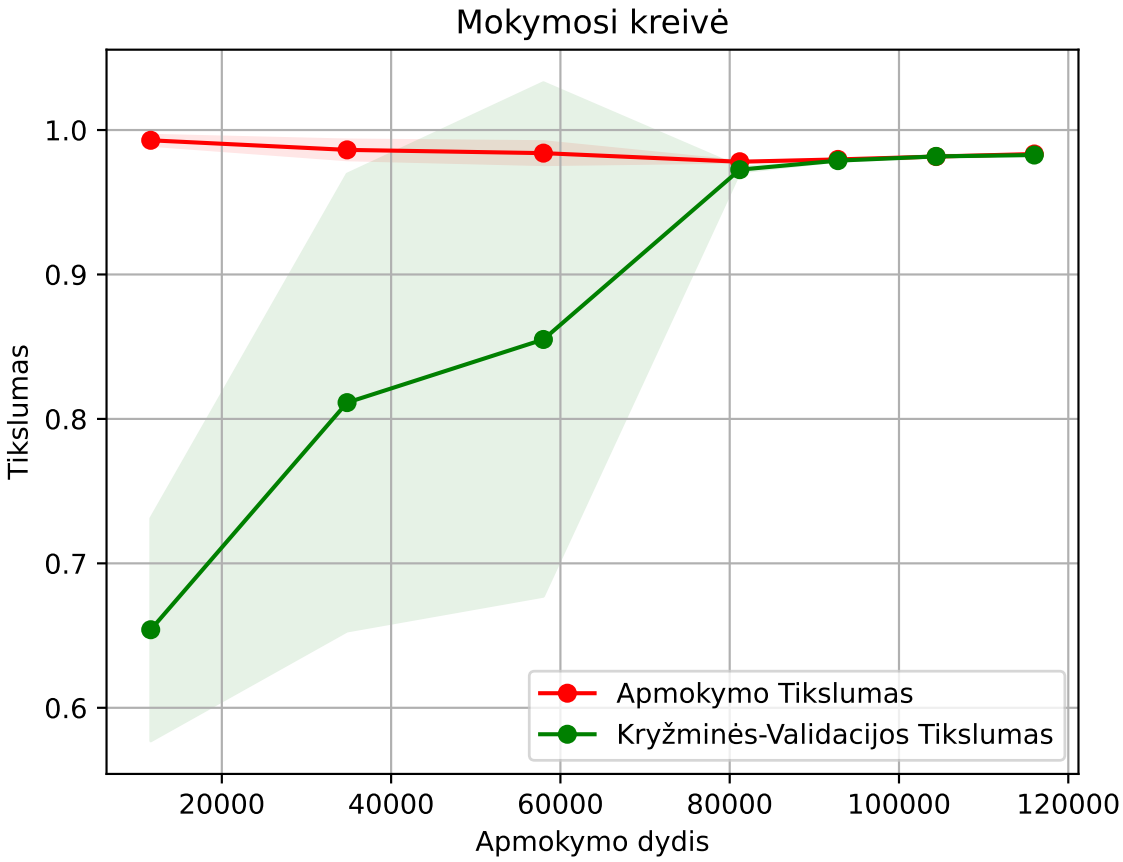
Pastebime, kad reikšmingo skirtumo tarp modelių nėra, tačiau matome, kad modelis toliau duoda šiek tiek geresnius rezultatus, todėl galime teigti, kad modelis nėra permokintas. Taip pat galima pabrėžti, kad duomenų rinkinys yra modifikuotas ir atsparus persimokinimui.

5.7.3 Mokymosi kreivė

Mokymosi kreivė yra įrankis mašininio mokymosi modelių persimokymui įvertinti. Ji suteikia informacijos apie modelio našumą didėjant mokymo duomenų dydžiui. Nubraižius mokymosi kreivę, galima stebėti mokymo ir kryžminio patikrinimo rezultatų tendenciją mokymo pavyzdžių skaičiaus atžvilgiu. Esant persimokymui, modelis turi tendenciją itin gerai dirbti su mokymo duomenimis, o su nematytais duomenimis jo našumas labai sumažėjimą. Tai rodo prastą apibendrinimą[10]. Mokymosi kreivė padeda nustatyti persimokymą atvaizduodama mokymo ir kryžminio patikrinimo rezultatų konvergenciją arba divergenciją. Jei tarp šių dviejų kreivių yra didelis atotrūkis, tai rodo, kad modelis per daug prisitaiko, o tai reiškia, kad modelis įsimena mokymo duomenis, o ne mokosi bendrų dėsningumų. Tokiais atvejais galima taikyti tokius metodus kaip reguliavimas, apkarpymas (angl. *Pruning*) arba mokymo duomenų dydžio didinimas, kad būtų sušvelnintas persimokymas ir pagerintas modelio apibendrinimo našumas.

Analizuojamas duomenų rinkinys: naudojamos apmokymo imties dydis atitinkamai –

10%, 30%, 50%, 70%, 80%, 90%, 100% (pav. 25).



25 pav.: Mokymo kreivė

Taip pat pridedama ir tikslumo sklaida, kai naudojamos imtys mažesnės ar lygios 50%, kadangi galima paimti kelias imtis. Pastebime, kad didinant duomenų kiekį, kryžminės validacijos tikslumas auga, jis beveik pilnai prisisotina naudojant 80% apmokymo imties, tačiau toliau didinant apmokymo imtį ji toliau auga, kas nėra keista, kadangi kuo daugiau duomenų turi modelis apmokymui, tuo dažniau, jis duoda geresnius rezultatus. Kartais galime pastebėti ir nukrypimą nuo didelio kiekio duomenų, kai modelis pradeda įsiminti atsakymus ir validacijos kreivė pradeda kristi žemyn, tačiau šio reiškinio nematome, todėl galime teigti, kad modelis nėra permokintas.

6 Rezultatai ir išvados

Bakalauro baigiamajame darbe atlikta mašininio mokymosi metodų analizė siekiant nuspėti telekomunikacijų įmonės klientų elgseną. Sprendžiant problemą pasitelkti trys klasifikacijos metodai: sprendimų medis, atsitiktiniai miškai ir LSTM algoritmas. Modelių tikslas buvo identifikuoti klientus, kurie atsisakys telekomunikacijų įmonės teikiamų paslaugų.

Pradinė duomenų analizė, tirianti duomenų tarpusavio ryšius parodė, Churn pasiskirstymą bei koreliacijas su Churn, naudojant *point − biserial* koreliacijos koeficientą. Stebėtas klientų pasiskirstymas, pagal prakalbėtą laiką, bei nagrinėtos išskirtys.

Pritaikytas prižiūrimas mašininis mokymasis, naudojant įvairius hiperparametrus. Su geriausiu gautu atsitiktinio miško modeliu toliau buvo atliekami įvairūs eksperimentai, kaip – reikšmingumo rodiklių analizė, nustatanti svarbiausias charakteristikas, pagal kurias galima identifikuoti Churn'erius, taip pat pritaikyta PCA analizė, siekiant pagerinti rezultatus, ir transformavus duomenis patikrinti ar reikšmingumo rodikliai kinta. Sprendimų medis ir atsitiktinių miškų algoritmas, su geriausiais parametrais, generavo patikimus rezultatus, iš kurių pavyko identifikuoti virš 80% Churn'erių ir teisingai nuspėti net 95% testinės imties duomenų. LSTM algoritmas generuodavo prastus rezultatus, teigdavo, kad visi klientai testuojamoje aibėje yra arba Churn'eriai, arba ne Churner'iai. Net stengiantis subalansuoti modelį, priskiriant Churn'eriams svorius, modelis teisingai identifikuodavo tik apie 10% Churn'erių, kadangi šis algoritmas labiau pritaikytas laike kintančių duomenų analizei.

Reikšminių atributų analizė parodo, kad beveik visi turimi kintamieji atneša panašią vertę siekiant identifikuoti Churn'erius. Atlikus PCA analizę geresni rezultatai nebuvo gauti, tačiau pastebėta, kad pakinta reikšminiai atributai, identifikuojantys Churn'erius. Pagal tai galime teigti, kad visi turimi duomenys yra vertingi ir pagerina prognozes.

Reikšmių pasiskirstymas imponuoja, kad gauti duomenys ar duomenų dalis yra pagaminti dirbtinai, greičiausiai taikant *Oversampling* metodą, dėl to neįmanoma kokybiškai atskirti testinės duomenų aibės. Gauti tikslumo rodikliai taip pat teigia, kad duomenys modifikuoti, kadangi dažnai dirbant su realiais duomenimis tokių tikslių rezultatų nepavyksta gauti.

Dėl duomenų ir gautų rezultatų specifiškumo, pastebėta, kad geriausias atsitiktinio miško modelis yra sudėtingas su neribotu gyliu ir minimaliais išskyrimo bei lapo sudarymo hiperparametrais. Nuspręsta papildomai atlikti *Overfitting* analizę geriausiai pasirodžiusiam modeliui –

atsitiktiniam miškui, siekiant išsiaiškinti ar modelis yra persimokęs, ar jį galima permokinti. Overfitting'o analizė parodė, kad modelio permokinti nepavyko. Didinant medžių skaičių, ar apmokymo imties dydį jis toliau generavo tokius pat gerus rezultatus, todėl galime teigti, kad gauti duomenys jau yra modifikuoti.

Literatūra

[1] Andrej Bugajev, Rima Kriauzienė, Olegas Vasilecas, Viktoras Chadyšas, The Impact of Churn Labelling Rules on Churn Prediction in Telecommunications, Informatica 33(2022), no. 2, 247-277, DOI 10.15388/22-INFOR484

[2] Churn prediction in telecommunication industry using kernel Support Vector Machines by Nguyen Nhu Y.ID, Tran Van LyID, Dao Vu Truong Son https://journals.plos.org/plosone/article/file?id=10.1371/journal.pone.0267935&type=printable

[3] Quinlan, J. R. (1986). Induction of decision trees. Machine learning, 1, 81-106.

[4] Prashant Gupta (2017) Decision Trees in Machine Learning https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052

[5] Stephanie Glen (2019) Decision Tree vs Random Forest vs Gradient Boosting Machines: Explained Simply https://www.datasciencecentral.com/decision-tree-vs-random-forest-vs-boosted-trees-explained/

[6] Alboukaey, N., Joukhadar, A., Ghneim, N. (2020). Dynamic behavior based churn prediction in mobile telecom. Expert Systems with Applications, 162, 113779.

[7] The Right Way to Oversample in Predictive Modeling https://beckernick.github.io/oversampling-modeling/

[8] Mitchell, Tom (1997). Machine Learning. New York: McGraw Hill.

[9] ODSC Community (2020) https://opendatascience.com/what-is-pruning-in-machine-learning/

[10] Baeldung (2023) https://www.baeldung.com/cs/learning-curve-ml

[11] V. Žalkauskas "Informatikos, kompiuterijos ir telekomunikacijų anglų-lietuvių kalbų žodynas"

A Priedas

```
# -*- coding: utf-8 -*-
"""Churn Analysis MU.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1-LTIDcZg_M4kHny2EQ_luxXLjyRDu4lA
"""

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns # For creating plots
import matplotlib.ticker as mtick # For specifying the axes tick format
import matplotlib.pyplot as plt
import scipy.stats as stats
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from sklearn.metrics import accuracy_score, f1_score, recall_score
from tensorflow.keras.layers import Embedding, Conv1D, AveragePooling1D,
                                    Bidirectional, LSTM, Dense
from tensorflow.keras.utils import plot_model
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, train_test_split,
                                    learning_curve
from imblearn.over_sampling import RandomOverSampler
from sklearn.tree import export_graphviz
from sklearn.decomposition import PCA
import graphviz


sns.set(style = 'white')

df = pd.read_csv('Call Details-Data.csv')
df.info()
```

```
df.head()

df.dtypes

df.isnull().sum()

# Replace True with 1 and False with 0
df['Churn'].replace(to_replace=True, value=1, inplace=True)
df['Churn'].replace(to_replace=False,  value=0, inplace=True)

# Plot the histogram
ax = df['Churn'].hist(bins=2)

# Get the counts for each value
counts = df['Churn'].value_counts()

# Add percentage labels to each bar of the histogram
for i, v in enumerate(counts):
    ax.text(abs(i-0.25), v/2, f'{v/len(df)*100:.1f}%', ha='center', va='center
                                            ', color='black', fontsize=12)

ax.set_title('Churn pasiskirstymas')
ax.set_ylabel('  moni   skai ius')
ax.set_xticks([0.25, 0.75])
ax.set_xticklabels(['Ne Churn', 'Churn'])
# Show the plot
plt.savefig('Churn_distribution.pdf')
plt.show()



# Read the data into a DataFrame
df = pd.read_csv('Call Details-Data.csv')  # Replace 'your_data.csv' with the
                                        actual file name or path

# Separate the binary and linear variables
binary_data = df['Churn']
```

```
linear_data = df.drop('Churn', axis=1).drop('Phone Number', axis = 1)  # Drop
                                        the 'Churn' column from the DataFrame

# Calculate the point-biserial correlation coefficient for each linear
                                        variable
correlations = {}
for column in linear_data:
    correlation, p_value = stats.pointbiserialr(binary_data, linear_data[
                                            column])
    correlations[column] = correlation

# Print the correlation coefficients
for column, correlation in correlations.items():
    print(f"Correlation between Churn and {column}: {correlation}")


# Sort the correlations in descending order
sorted_correlations = {k: v for k, v in sorted(correlations.items(), key=
                                        lambda item: item[1], reverse=True)}

# Create a bar chart of the correlation coefficients

plt.figure(figsize=(12, 8))
bar_colors = ['skyblue' if c > 0 else 'lightcoral' for c in
                                        sorted_correlations.values()]
plt.bar(sorted_correlations.keys(), sorted_correlations.values(), color=
                                        bar_colors)
plt.xlabel('Tiesiniai kintamieji', fontsize=12)
plt.ylabel('Koreliacijos koeficientai', fontsize=12)
plt.title('Koreliacijos su Churn', fontsize=14)
plt.xticks(rotation=45, ha='right', fontsize=10)
plt.yticks(fontsize=10)

# Add data labels to the bars
for i, (x, y) in enumerate(zip(sorted_correlations.keys(), sorted_correlations
                                        .values())):
    plt.text(i, y, f'{y:.2f}', ha='center', va='bottom', fontsize=10)
```

```
plt.grid(axis='y', linestyle='--', alpha=0.5)  # Add horizontal gridlines
plt.tight_layout()
plt.savefig('Koreliacijos.pdf')
plt.show()

sns.histplot(df['Intl Mins'], bins=10)   # plot a histogram of a variable to
                                          visualize its distribution
#sns.boxplot(y=df['VMail Message'])   # plot a boxplot of a variable to
                                          visualize its distribution and outliers

sns.scatterplot(x='Day Mins', y='Eve Mins', data=df)   # plot a scatterplot to
                                           visualize the relationship between two
                                           variables
plt.savefig('Day-eve mins scattter.pdf')
plt.show()

N = len(df['Day Mins'])
plt.figure(figsize=(8, 6)) # set figure size
plt.style.use('seaborn-whitegrid') # set style
plt.yscale("log")
plt.plot(range(N), df['Day Mins'].sort_values(), linewidth=2) # increase line
                                          width

plt.xlabel('  moni   skai ius', fontsize=14) # increase font size
plt.ylabel('Dien  prakalb tos minut s', fontsize=14) # increase font size
plt.title('Logaritminis   moni   pasiskirstymas pagal dien  prakalb tas
                                          minutes', fontsize=16) # increase font
                                          size
plt.xticks(fontsize=12) # increase x-axis tick font size
plt.yticks(fontsize=12) # increase y-axis tick font size

plt.savefig('Logaritminis   moni   pasiskirstymas pagal dien  prakalb tas
                                          minutes.pdf')
plt.show()

#ilgai krauna (4min mazdaug)
sns.pairplot(df, vars=['Day Mins', 'Eve Mins', 'Night Mins'])# plot a pairwise
                                           scatterplot to visualize the
```

```
                                        relationships between multiple
                                        variables
plt.savefig('Day_Eve_Night Mins scatter.pdf')
plt.show()

plt.figure(figsize=(10, 8))
corr_matrix = df.drop('Churn', axis=1).corr().round(1)
sns.heatmap(corr_matrix, annot=True)
plt.savefig('Koreliaciju matrica.pdf')
plt.show()

from scipy import stats
# Define the columns to analyze for outliers
cols = ['Account Length', 'VMail Message', 'Day Mins', 'Day Calls', 'Day
                                        Charge',
        'Eve Mins', 'Eve Calls', 'Eve Charge', 'Night Mins', 'Night Calls',
        'Night Charge', 'Intl Mins', 'Intl Calls', 'Intl Charge', 'CustServ
                                            Calls']

# Create boxplots for each column to detect outliers
z = np.abs(stats.zscore(df[cols]))

# Define the threshold for outlier detection
threshold = 1.96

# Create a boolean array indicating whether each data point is an outlier
outliers = (z > threshold).any(axis=1)

# Print the number of outliers and the indices of the outliers
print(f"Number of outliers: {outliers.sum()}")
print(f"Indices of outliers: {np.where(outliers)[0]}")
print("total number of users:",len(df))

# Calculate the IQR for each column
Q1 = df[cols].quantile(0.25)
Q3 = df[cols].quantile(0.75)
IQR = Q3 - Q1
```

```
# Define the threshold for outlier detection
threshold = 1.5

# Create a boolean array indicating whether each data point is an outlier
outliers = ((df[cols] < (Q1 - threshold * IQR)) | (df[cols] > (Q3 + threshold
                                        * IQR))).any(axis=1)

# Print the number of outliers and the indices of the outliers
print(f"Number of outliers: {outliers.sum()}")
print(f"Indices of outliers: {np.where(outliers)[0]}")

z = np.abs(stats.zscore(df[cols]))

# Define the threshold for outlier detection
threshold = 3

# Create a boolean array indicating whether each data point is an outlier
outliers = (z > threshold).any(axis=1)

# Print the number of outliers and the indices of the outliers
print(f"Number of outliers: {outliers.sum()}")
print(f"Indices of outliers: {np.where(outliers)[0]}")

z = np.abs(stats.zscore(df[cols]))

# Define the threshold for outlier detection
threshold = 3

# Create a boolean array indicating whether each data point is an outlier
outliers = (z > threshold).any(axis=1)

outliers_df = df[outliers]
print(outliers_df)
outliers_df["Churn"].sum()

df = df.sample(frac=1, random_state=42)

# Preprocess the data
```

```
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(df.iloc[:, 1:-1])

# Define the number of time steps and features
timesteps = 1
features = scaled_data.shape[1]

# Add the churn column to the preprocessed data
scaled_data = np.concatenate([scaled_data, np.array(df['Churn']).reshape(-1,1)
                                          ], axis=1)

# Split the data into training and validation sets
train_size = int(len(df) * 0.8)
train_data = scaled_data[:train_size, :]
val_data = scaled_data[train_size:, :]

# Reshape the data for the LSTM model
trainX, trainY = [], []
for i in range(timesteps, train_data.shape[0]):
    trainX.append(train_data[i-timesteps:i, :-1])
    trainY.append(train_data[i, -1])
trainX, trainY = np.array(trainX), np.array(trainY)

valX, valY = [], []
for i in range(timesteps, val_data.shape[0]):
    valX.append(val_data[i-timesteps:i, :-1])
    valY.append(val_data[i, -1])
valX, valY = np.array(valX), np.array(valY)

# Build the LSTM model
model = Sequential()
model.add(LSTM(200, return_sequences=True, input_shape=(trainX.shape[1],
                                          features)))
model.add(LSTM(100))
model.add(Dense(1))

# Compile the model
```

```
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'
                                        ])

# Train the model
history = model.fit(trainX, trainY, epochs=10, batch_size=64, validation_data=
                                        (valX, valY), verbose=1)

# Evaluate the model
train_pred = model.predict(trainX)
train_pred = np.round(train_pred)
train_acc = accuracy_score(trainY, train_pred)
train_f1 = f1_score(trainY, train_pred)
train_recall = recall_score(trainY, train_pred)

val_pred = model.predict(valX)
val_pred = np.round(val_pred)
val_acc = accuracy_score(valY, val_pred)
val_f1 = f1_score(valY, val_pred)
val_recall = recall_score(valY, val_pred)

print("Training accuracy: {:.2f}%".format(train_acc*100))
print("Training F1-score: {:.2f}%".format(train_f1*100))
print("Training Recall: {:.2f}%".format(train_recall*100))
print("Validation accuracy: {:.2f}%".format(val_acc*100))
print("Validation F1-score: {:.2f}%".format(val_f1*100))
print("Validation Recall: {:.2f}%".format(val_recall*100))

total_samples = len(trainY)
n_churn_samples = np.sum(trainY)
n_non_churn_samples = total_samples - n_churn_samples
churn_weight = n_non_churn_samples / n_churn_samples
class_weight = {0: 1.0, 1: churn_weight}
print(churn_weight)

# Build the LSTM model
model = Sequential()
model.add(LSTM(200, return_sequences=True, input_shape=(trainX.shape[1],
                                        features)))
```

```
model.add(LSTM(100))
model.add(Dense(1))

# Compile the model
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'
                                        ])

# Train the model
history = model.fit(trainX, trainY, epochs=10, batch_size=64, validation_data=
                                        (valX, valY), verbose=1, class_weight=
                                        class_weight)

# Evaluate the model
train_pred = model.predict(trainX)
train_pred = np.round(train_pred)
train_acc = accuracy_score(trainY, train_pred)
train_f1 = f1_score(trainY, train_pred)
train_recall = recall_score(trainY, train_pred)

val_pred = model.predict(valX)
val_pred = np.round(val_pred)
val_acc = accuracy_score(valY, val_pred)
val_f1 = f1_score(valY, val_pred)
val_recall = recall_score(valY, val_pred)

print("Training accuracy: {:.2f}%".format(train_acc*100))
print("Training F1-score: {:.2f}%".format(train_f1*100))
print("Training Recall: {:.2f}%".format(train_recall*100))
print("Validation accuracy: {:.2f}%".format(val_acc*100))
print("Validation F1-score: {:.2f}%".format(val_f1*100))
print("Validation Recall: {:.2f}%".format(val_recall*100))

# Load dataset
df = pd.read_csv('Call Details-Data.csv')

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
```

```
    df.drop('Churn', axis=1).drop('Phone Number', axis = 1), df['Churn'],
                                                    test_size=0.2, random_state=42)

# Apply oversampling to training data
ros = RandomOverSampler(random_state=42)
X_train, y_train = ros.fit_resample(X_train, y_train)

# Define parameter grid for GridSearchCV
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [5, 10, 20, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initialize and fit GridSearchCV with RandomForestClassifier
rf = RandomForestClassifier(random_state=42)
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, verbose
                                                = 1)
grid_search.fit(X_train, y_train)

# Print best parameters and best score
print('Best parameters:', grid_search.best_params_)
print('Best score:', grid_search.best_score_)

# Evaluate model on test set
y_pred = grid_search.predict(X_test)
acc = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
print('Test accuracy:', acc)
print('Test F1 score:', f1)
print('Test recall:', recall)

# Load dataset
df = pd.read_csv('Call Details-Data.csv')

# Split data into train and test sets
```

```
X_train, X_test, y_train, y_test = train_test_split(
    df.drop('Churn', axis=1).drop('Phone Number', axis = 1), df['Churn'],
                                                test_size=0.2, random_state=42)

# Apply oversampling to training data
ros = RandomOverSampler(random_state=42)
X_train, y_train = ros.fit_resample(X_train, y_train)

# Define parameter grid for GridSearchCV
param_grid = {
    'n_estimators': [100],
    'max_depth': [None],
    'min_samples_split': [2],
    'min_samples_leaf': [1]
}

# Initialize and fit GridSearchCV with RandomForestClassifier
rf = RandomForestClassifier(random_state=42)
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, verbose
                                            = 1)
grid_search.fit(X_train, y_train)

# Print best parameters and best score
print('Best parameters:', grid_search.best_params_)
print('Best score:', grid_search.best_score_)

# Evaluate model on test set
y_pred = grid_search.predict(X_test)
acc = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
print('Test accuracy:', acc)
print('Test F1 score:', f1)
print('Test recall:', recall)

# Get feature importances
importances = grid_search.best_estimator_.feature_importances_
```

```
# Create a dataframe to store feature importances
feature_importances = pd.DataFrame({'feature': X_train.columns, 'importance':
                                    importances})

# Sort features by importance
feature_importances = feature_importances.sort_values('importance', ascending=
                                    True)

# Print feature importances
print(feature_importances)

plt.figure(figsize=(10, 8))  # Set the desired figure size
plt.barh(feature_importances.feature, feature_importances.importance)
plt.xlabel('Svarba')
plt.ylabel('Kintamieji')
plt.title('Reik mingumo rodikliai')
plt.tight_layout()  # Adjust the spacing between subplots and labels
plt.savefig('Svarbiausi_kriterijai.pdf')

# Load dataset
df = pd.read_csv('Call Details-Data.csv')

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    df.drop('Churn', axis=1).drop('Phone Number', axis = 1), df['Churn'],
                                        test_size=0.2, random_state=42)

# Apply oversampling to training data
ros = RandomOverSampler(random_state=42)
X_train, y_train = ros.fit_resample(X_train, y_train)

# Define parameter grid for GridSearchCV
param_grid = {
    'n_estimators': [200],
    'max_depth': [None],
    'min_samples_split': [2],
    'min_samples_leaf': [1]
}
```

```
# Initialize and fit GridSearchCV with RandomForestClassifier
rf = RandomForestClassifier(random_state=42)
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, verbose
                                        = 1)
grid_search.fit(X_train, y_train)

# Print best parameters and best score
print('Best parameters:', grid_search.best_params_)
print('Best score:', grid_search.best_score_)

# Evaluate model on test set
y_pred = grid_search.predict(X_test)
acc = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
print('Test accuracy:', acc)
print('Test F1 score:', f1)
print('Test recall:', recall)

# Load dataset
df = pd.read_csv('Call Details-Data.csv')

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    df.drop('Churn', axis=1).drop('Phone Number', axis = 1), df['Churn'],
                                            test_size=0.2, random_state=42)

# Apply oversampling to training data
ros = RandomOverSampler(random_state=42)
X_train, y_train = ros.fit_resample(X_train, y_train)

# Define parameter grid for GridSearchCV
param_grid = {
    'n_estimators': [1000],
    'max_depth': [None],
    'min_samples_split': [2],
    'min_samples_leaf': [1]
```

```
}

# Initialize and fit GridSearchCV with RandomForestClassifier
rf = RandomForestClassifier(random_state=42)
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, verbose
                                        = 1)
grid_search.fit(X_train, y_train)

# Print best parameters and best score
print('Best parameters:', grid_search.best_params_)
print('Best score:', grid_search.best_score_)

# Evaluate model on test set
y_pred = grid_search.predict(X_test)
acc = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
print('Test accuracy:', acc)
print('Test F1 score:', f1)
print('Test recall:', recall)

# Load dataset
df = pd.read_csv('Call Details-Data.csv')

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    df.drop('Churn', axis=1).drop('Phone Number', axis = 1), df['Churn'],
                                            test_size=0.2, random_state=42)

# Initialize and fit decision tree classifier
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)

# Evaluate model on test set
y_pred = dt.predict(X_test)
acc = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
```

```
# Calculate number of leaf nodes
leaf_nodes = sum([1 for val in dt.tree_.value if val.any()])

print('Test accuracy:', acc)
print('Test F1 score:', f1)
print('Test recall:', recall)
print('Number of leaf nodes:', leaf_nodes)

# Export decision tree to DOT format
dot_data = export_graphviz(dt, out_file=None,
                           feature_names=X_train.columns,
                           class_names=['No Churn', 'Churn'],
                           filled=True, rounded=True,
                           special_characters=True)

# Visualize decision tree with Graphviz
graph = graphviz.Source(dot_data)
graph.format = 'pdf'
graph.render('decision_tree')

# Load dataset
df = pd.read_csv('Call Details-Data.csv')

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    df.drop('Churn', axis=1).drop('Phone Number', axis = 1), df['Churn'],
                                            test_size=0.2, random_state=42)

# Initialize and fit decision tree classifier with max depth of 10
dt = DecisionTreeClassifier(max_depth=10, random_state=42)
dt.fit(X_train, y_train)

# Evaluate model on test set
y_pred = dt.predict(X_test)
acc = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
```

```
leaf_nodes = sum([1 for val in dt.tree_.value if val.any()])

print('Test accuracy:', acc)
print('Test F1 score:', f1)
print('Test recall:', recall)
print('Number of leaf nodes:', leaf_nodes)

plt.title('Modelio tikslumas')
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.ylabel('Tikslumas')
plt.xlabel('epoch  skai ius')
plt.legend(['mokymo','validavimo'])
plt.savefig('LSTM_weighted_tikslumas.pdf')
plt.show()
#---------------------------------------
plt.plot(history.history['loss'])

plt.title('Modelio paklaidos funkcija')
plt.plot(history.history['val_loss'])
plt.xlabel('epoch  skai ius')
plt.ylabel('Paklaida')
plt.legend(['mokymo','validavimo'])
plt.savefig('LSTM_weighted_loss.pdf')
plt.show()

# Load dataset
df = pd.read_csv('Call Details-Data.csv')
df['Mins'] = df['Day Mins'] + df['Eve Mins'] + df['Night Mins']
df['Calls'] = df['Day Calls'] + df['Eve Calls'] + df['Night Calls']
df['Charge'] = df['Day Charge'] + df['Eve Charge'] + df['Night Charge']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    df.drop('Churn', axis=1).drop('Phone Number', axis = 1), df['Churn'],
                                            test_size=0.2, random_state=42)
```

```
# Apply oversampling to training data
ros = RandomOverSampler(random_state=42)
X_train, y_train = ros.fit_resample(X_train, y_train)

# Define parameter grid for GridSearchCV
param_grid = {
    'n_estimators': [100],
    'max_depth': [None],
    'min_samples_split': [2],
    'min_samples_leaf': [1]
}

# Initialize and fit GridSearchCV with RandomForestClassifier
rf = RandomForestClassifier(random_state=42)
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, verbose
                                        = 1)
grid_search.fit(X_train, y_train)

# Print best parameters and best score
print('Best parameters:', grid_search.best_params_)
print('Best score:', grid_search.best_score_)

# Evaluate model on test set
y_pred = grid_search.predict(X_test)
acc = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
print('Test accuracy:', acc)
print('Test F1 score:', f1)
print('Test recall:', recall)

# Get feature importances
importances = grid_search.best_estimator_.feature_importances_

# Create a dataframe to store feature importances
feature_importances = pd.DataFrame({'feature': X_train.columns, 'importance':
                                        importances})
```

```
# Sort features by importance
feature_importances = feature_importances.sort_values('importance', ascending=
                                                    True)

# Print feature importances
print(feature_importances)

plt.figure(figsize=(10, 8))  # Set the desired figure size
plt.barh(feature_importances.feature, feature_importances.importance)
plt.xlabel('Svarba')
plt.ylabel('Kintamieji')
plt.title('Reik mingumo rodikliai')
plt.tight_layout()  # Adjust the spacing between subplots and labels
plt.savefig('Svarbiausi_kriterijai + aggreguoti.pdf')

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    df.drop('Churn', axis=1).drop('Phone Number', axis = 1).drop('Day Mins',
                                                    axis = 1).drop('Eve Mins', axis = 1
                                                    ).drop('Night Mins', axis = 1)
    .drop('Day Calls', axis = 1).drop('Eve Calls', axis = 1).drop('Night Calls
                                                    ', axis = 1)
    .drop('Day Charge', axis = 1).drop('Eve Charge', axis = 1).drop('Night
                                                    Charge', axis = 1), df['Churn'],
                                                    test_size=0.2, random_state=42)

# Apply oversampling to training data
ros = RandomOverSampler(random_state=42)
X_train, y_train = ros.fit_resample(X_train, y_train)

# Define parameter grid for GridSearchCV
param_grid = {
    'n_estimators': [100],
    'max_depth': [None],
    'min_samples_split': [2],
    'min_samples_leaf': [1]
}
```

```
# Initialize and fit GridSearchCV with RandomForestClassifier
rf = RandomForestClassifier(random_state=42)
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, verbose
                                        = 1)
grid_search.fit(X_train, y_train)

# Print best parameters and best score
print('Best parameters:', grid_search.best_params_)
print('Best score:', grid_search.best_score_)

# Evaluate model on test set
y_pred = grid_search.predict(X_test)
acc = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
print('Test accuracy:', acc)
print('Test F1 score:', f1)
print('Test recall:', recall)

# Load dataset
df = pd.read_csv('Call Details-Data.csv')

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    df.drop(['Churn', 'Phone Number'], axis=1), df['Churn'], test_size=0.2,
                                            random_state=42)

# Apply over-sampling to training data
ros = RandomOverSampler(random_state=42)
X_train, y_train = ros.fit_resample(X_train, y_train)

# Define the desired train_sizes as relative sizes
train_sizes = [0.1, 0.3, 0.5, 0.7, 0.8, 0.9, 1.0]

# Define parameter grid for GridSearchCV
param_grid = {
    'n_estimators': [100],
    'max_depth': [None],
```

```
    'min_samples_split': [2],
    'min_samples_leaf': [1]
}

# Initialize and fit GridSearchCV with RandomForestClassifier
rf = RandomForestClassifier(random_state=42)
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, verbose=
                                        1)
grid_search.fit(X_train, y_train)

# Print best parameters and best score
print('Best parameters:', grid_search.best_params_)
print('Best score:', grid_search.best_score_)

# Evaluate model on test set
y_pred = grid_search.predict(X_test)
acc = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
print('Test accuracy:', acc)
print('Test F1 score:', f1)
print('Test recall:', recall)

# Plot learning curve
train_sizes, train_scores, test_scores = learning_curve(
    grid_search.best_estimator_, X_train, y_train, cv=5, scoring='accuracy',
                                            train_sizes=train_sizes, n_jobs=-1)

train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)

plt.figure()
plt.title('Mokymosi kreiv ')
plt.xlabel('Apmokymo dydis')
plt.ylabel('Tikslumas')
plt.grid()
```

```
plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                 train_scores_mean + train_scores_std, alpha=0.1, color='r')
plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                 test_scores_mean + test_scores_std, alpha=0.1, color='g')
plt.plot(train_sizes, train_scores_mean, 'o-', color='r', label='Apmokymo
                                        Tikslumas')
plt.plot(train_sizes, test_scores_mean, 'o-', color='g', label='Kry min s -
                                        Validacijos Tikslumas')

plt.legend(loc='lower right')
plt.savefig('Learning curve.pdf')
plt.show()

# Load dataset
df = pd.read_csv('Call Details-Data.csv')
df['Mins'] = df['Day Mins'] + df['Eve Mins'] + df['Night Mins']
df['Calls'] = df['Day Calls'] + df['Eve Calls'] + df['Night Calls']
df['Charge'] = df['Day Charge'] + df['Eve Charge'] + df['Night Charge']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    df.drop('Churn', axis=1).drop('Phone Number', axis=1).drop('Day Mins',
                                        axis=1).drop('Eve Mins', axis=1).
                                        drop(
        'Night Mins', axis=1)
    .drop('Day Calls', axis=1).drop('Eve Calls', axis=1).drop('Night Calls',
                                        axis=1)
    .drop('Day Charge', axis=1).drop('Eve Charge', axis=1).drop('Night Charge'
                                        , axis=1), df['Churn'], test_size=0
                                        .2,
    random_state=42)

# Apply oversampling to training data
ros = RandomOverSampler(random_state=42)
X_train, y_train = ros.fit_resample(X_train, y_train)

# Define parameter grid for GridSearchCV
```

```
param_grid = {
    'n_estimators': [100],
    'max_depth': [None],
    'min_samples_split': [2],
    'min_samples_leaf': [1]
}

# Initialize and fit GridSearchCV with RandomForestClassifier
rf = RandomForestClassifier(random_state=42)
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, verbose
                                        = 1)
grid_search.fit(X_train, y_train)

# Print best parameters and best score
print('Best parameters:', grid_search.best_params_)
print('Best score:', grid_search.best_score_)

# Evaluate model on test set
y_pred = grid_search.predict(X_test)
acc = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
print('Test accuracy:', acc)
print('Test F1 score:', f1)
print('Test recall:', recall)



# Get feature importances
importances = grid_search.best_estimator_.feature_importances_

# Create a dataframe to store feature importances
feature_importances = pd.DataFrame({'feature': X_train.columns, 'importance':
                                        importances})

# Sort features by importance
feature_importances = feature_importances.sort_values('importance', ascending=
                                        True)
```

```
# Print feature importances
print(feature_importances)

plt.figure(figsize=(10, 8))  # Set the desired figure size
plt.barh(feature_importances.feature, feature_importances.importance)
plt.xlabel('Svarba')
plt.ylabel('Kintamieji')
plt.title('Reik mingumo rodikliai')
plt.tight_layout()  # Adjust the spacing between subplots and labels
#plt.savefig('Svarbiausi_kriterijai - tik agreguoti.pdf')

# Load dataset
df = pd.read_csv('Call Details-Data.csv')

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    df.drop(['Churn', 'Phone Number'], axis=1), df['Churn'], test_size=0.2,
                                            random_state=42)

# Apply over-sampling to training data
ros = RandomOverSampler(random_state=42)
X_train, y_train = ros.fit_resample(X_train, y_train)

# Perform PCA
n_components = 10  # Set the desired number of PCA components
pca = PCA(n_components=n_components)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

# Define parameter grid for GridSearchCV
param_grid = {
    'n_estimators': [100],
    'max_depth': [None],
    'min_samples_split': [2],
    'min_samples_leaf': [1]
}
```

```
# Initialize and fit GridSearchCV with RandomForestClassifier
rf = RandomForestClassifier(random_state=42)
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, verbose=
                                        1)
grid_search.fit(X_train_pca, y_train)

# Print best parameters and best score
print('Best parameters:', grid_search.best_params_)
print('Best score:', grid_search.best_score_)

# Evaluate model on test set
y_pred = grid_search.predict(X_test_pca)
acc = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
print('Test accuracy:', acc)
print('Test F1 score:', f1)
print('Test recall:', recall)

# Get PCA component composition
component_composition = pca.components_

# Get original feature names
original_feature_names = df.drop(['Churn', 'Phone Number'], axis=1).columns

# Create a DataFrame for component composition
component_composition_df = pd.DataFrame(component_composition.T, columns=['PC{
                                        }'.format(i+1) for i in range(
                                        n_components)], index=
                                        original_feature_names)

# Print PCA component composition
print('PCA Component Composition:')
print(component_composition_df)

# Load dataset
df = pd.read_csv('Call Details-Data.csv')
```

```
# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    df.drop(['Churn', 'Phone Number'], axis=1), df['Churn'], test_size=0.2,
                                                random_state=42)

# Apply over-sampling to training data
ros = RandomOverSampler(random_state=42)
X_train, y_train = ros.fit_resample(X_train, y_train)

# Perform PCA
n_components = 5  # Set the desired number of PCA components
pca = PCA(n_components=n_components)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

# Define parameter grid for GridSearchCV
param_grid = {
    'n_estimators': [100],
    'max_depth': [None],
    'min_samples_split': [2],
    'min_samples_leaf': [1]
}

# Initialize and fit GridSearchCV with RandomForestClassifier
rf = RandomForestClassifier(random_state=42)
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, verbose=
                                            1)
grid_search.fit(X_train_pca, y_train)

# Print best parameters and best score
print('Best parameters:', grid_search.best_params_)
print('Best score:', grid_search.best_score_)

# Evaluate model on test set
y_pred = grid_search.predict(X_test_pca)
acc = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
```

```
print('Test accuracy:', acc)
print('Test F1 score:', f1)
print('Test recall:', recall)

# Get feature importances
importances = grid_search.best_estimator_.feature_importances_

# Get original feature names based on selected PCA components
original_feature_names = ['PC{}'.format(i+1) for i in range(n_components)]

# Create a DataFrame for feature importance
feature_importance_df = pd.DataFrame({'Feature': original_feature_names, '
                                      Importance': importances})

# Sort by importance (descending order)
feature_importance_df = feature_importance_df.sort_values(by='Importance',
                                      ascending=True)

# Get PCA component composition
component_composition = pca.components_

# Get original feature names
original_feature_names = df.drop(['Churn', 'Phone Number'], axis=1).columns

# Create a DataFrame for component composition
component_composition_df = pd.DataFrame(component_composition.T, columns=['PC{
                                      }'.format(i+1) for i in range(
                                      n_components)], index=
                                      original_feature_names)

# Print feature importance
print('Feature Importance:')
print(feature_importance_df)

plt.figure(figsize=(10, 4))  # Set the desired figure size
plt.barh(feature_importance_df.iloc[:, 0], feature_importance_df.iloc[:, 1])
plt.xlabel('Svarba')
plt.ylabel('Kintamieji')
```

```
plt.title('Reik mingumo rodikliai')
plt.tight_layout()  # Adjust the spacing between subplots and labels
plt.savefig('PCA5_Svarbiausi_kriterijai.pdf')

# Print PCA component composition
print('PCA Component Composition:')
print(component_composition_df)

# Load dataset
df = pd.read_csv('Call Details-Data.csv')

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    df.drop(['Churn', 'Phone Number'], axis=1), df['Churn'], test_size=0.2,
                                            random_state=42)

# Apply over-sampling to training data
ros = RandomOverSampler(random_state=42)
X_train, y_train = ros.fit_resample(X_train, y_train)

# Perform PCA
n_components = 15  # Set the desired number of PCA components
pca = PCA(n_components=n_components)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

# Define parameter grid for GridSearchCV
param_grid = {
    'n_estimators': [100],
    'max_depth': [None],
    'min_samples_split': [2],
    'min_samples_leaf': [1]
}

# Initialize and fit GridSearchCV with RandomForestClassifier
rf = RandomForestClassifier(random_state=42)
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, verbose=
                                        1)
```

```
grid_search.fit(X_train_pca, y_train)

# Print best parameters and best score
print('Best parameters:', grid_search.best_params_)
print('Best score:', grid_search.best_score_)

# Evaluate model on test set
y_pred = grid_search.predict(X_test_pca)
acc = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
print('Test accuracy:', acc)
print('Test F1 score:', f1)
print('Test recall:', recall)

# Get feature importances
importances = grid_search.best_estimator_.feature_importances_

# Get original feature names based on selected PCA components
original_feature_names = ['PC{}'.format(i+1) for i in range(n_components)]

# Create a DataFrame for feature importance
feature_importance_df = pd.DataFrame({'Feature': original_feature_names, '
                                      Importance': importances})

# Sort by importance (descending order)
feature_importance_df = feature_importance_df.sort_values(by='Importance',
                                      ascending=True)

# Get PCA component composition
component_composition = pca.components_

# Get original feature names
original_feature_names = df.drop(['Churn', 'Phone Number'], axis=1).columns

# Create a DataFrame for component composition
component_composition_df = pd.DataFrame(component_composition.T, columns=['PC{
                                      }'.format(i+1) for i in range(
```

```
                                        n_components)], index=
                                        original_feature_names)

# Print feature importance
print('Feature Importance:')
print(feature_importance_df)

plt.figure(figsize=(10, 8))  # Set the desired figure size
plt.barh(feature_importance_df.iloc[:, 0], feature_importance_df.iloc[:, 1])
plt.xlabel('Svarba')
plt.ylabel('Kintamieji')
plt.title('Reik mingumo rodikliai')
plt.tight_layout()  # Adjust the spacing between subplots and labels
plt.savefig('PCA15_Svarbiausi_kriterijai.pdf')


# Print PCA component composition
print('PCA Component Composition:')
print(component_composition_df)
```