LATVIJAS UNIVERSITĀTES
DATORIKAS FAKULTĀTE

PAPILDU FUNKCIJU IZSTRĀDE MAŠĪNMĀCĪŠANĀS TEKSTA ANALIZATORAM

KVALIFIKĀCIJAS DARBS

Autors: Mikus Kalniņš
Studenta apliecības numurs: mk17110
Darba vadītājs: prof. Jānis Zuters

RĪGA 2019

ANOTĀCIJA

Teksta analīzes veikšanai ir pieejamas vairākas metodes, šajā dokumentā ir apskatīts kā pielietot *Correlation Explanation*[7] (korelāciju skaidrošana, turpmāk *CorEx*[7]) teksta analīzes metodi, implementējot atvērtā pirmkoda (*open-source*) bibliotēku *corextopic*[3], jau izstrādātā sistēmā - mašīnmācīšanās teksta analizatorā (turpmāk *MMTA*). Darbā ir aprakstīta *MMTA* pamatdarbība un tā mijiedarbība ar implementēto *corextopic*[3] bibliotēku.

MMTA ir programmprodukts ar implementētām vairākām bibliotēkām, kas dod iespēju programmprodukta lietotājam izvēlēties starp vairākām datu apstrādes metodēm saistībā ar teksta analīzi. *MMTA* darbība iedalās. Informācijas ekstrakcija un tēmu modelēšana ir divi *MMTA* darbības iedalījumu piemēri. Darbā tiek apskatīta tēmu modelēšana, jo *corextopic*[3] bibliotēka sniedz bagātīgi modelētas tēmas, tā raksturojot apstrādājamo datu kopumu.

Atslēgvārdi: CorEx, tēmu modelēšana, mašīnmācīšanās, teksta analīze, python

ABSTRACT

Additional Feature Development for Machine Learning Text Analyzer

Text analysis has many algorithms that try to solve the problem of topic modeling. This document explains how the text analisys method called *Correlation Explanation*[7] was intergrated into a *Machine Learning Text Analyzer* (*MaLTA*) using an open-source *Python* library *corextopic*[3]. The document describes core functions of *MaLTA* and how they interact with the integrated *corextopic*[3] library.

MaLTA is a prgram product with many integrated open-source libraries, that offer the user of *MaLTA* to choose from a variety of methods to use for text analisys. There are several categories of text analysis that *MaLTA* works with. *Topic Modeling* and *Information Extraction* are two examples. The document describes the process of Topic Modeling since *corextopic*[3] offers richly modeled topics in such a way describing the input data.

Key-words: CorEx, topic modeling, machine learning, text analysis, Python

SATURA RĀDĪTĀJS

TERMINU SKAIDROJUMS

Atslēga – programmēšanas vērtības nosaukums, ko angliski sauc par *key*, kas ir daļa no *dictionary* (vārdnīcas).

Bibliotēka – rīks, iepriekš izveidots kods, ar izstrādei lietderīgām funkcijām.

Būla tips – patiesuma vērtība. Jā vai nē, 0 vai 1, Patiess vai nepatiess. Programmēšanā parasti atsaucas uz vērtībām "True" un "False" vai 0 un 1.

CorEx – *Correlation Explanation*[7] no agļu valodas tulkojot: korēlāciju skaidrošana.

Celmošana – teksta apstrādes process, kas apstrādā vārdus tā, lai no tiem paliktu tikai vārda sakne un nominatīvs. Bieži vien noņemot galotnes un pārveidojot vārdus vienskaitļa formā. Piemēram, vārds "Trouble" tiek pārvērst par "Troubl".

Datu tips – datu paveids. Piemēram, skaitlisks, simbolu virkne, simbols.

Inicializē – izveido, iesāk darbību.

Instance – objekts, objekta gadījums, esamība, objekts pats. Piemēram, monēta var būt ideja, taču monētas instance varētu būt kāda konkrēta monētā, kas tiek turēta rokā.

Iterēt – veikt kādu darbību atkārtoti, secīgi.

Klase – programmēšanas procesā izveidtos datu objekts, kas var saturēt funkcijas, vērtības. Veidne.

Konsole – vadības saskarne.

Lemmatizācija – līdzīgs process kā celmošana.

MMTA – saīsinājums no "Mašīnmācīšanās teksta analizators".

Nebūtisks vārds – vārds, kas dod maz inofrmācijas par teksta kopējo nozīmi, vai ir ar ļoti plašu nozīmi. Piemēram, "es, tu, mēs, grib, vienreiz". Pretējs piemērs, "dators, sāp, galvas sāpes, armija".

Noklusējuma – iepriekš izvēlēta vai uzstādīta vērtība.

PPA – Programmatūras projektējuma apraksts.

PPS – Programmatūras prasību specifikācija.

Teksta dokuments – viens teksta vienums. Piemēram, e-pasta saturs, grāmatas virsrakts, dziesmas vārdi. Šādi teksta dokumenti veido teksta dokumentu kopu.

Tokenizēšana – teksta analīzes sfērā apzīmē procesu, kas, sadala teikumu tokenos.

Tokens – teksta anlīzes sfērā ar apzīmē vārdu, vai vārda sakni.

Trenēt – mašīnmācīšanās termins, kas apzīmē procesu, kad iteratīvi modeli trenē uz kādiem datiem, lai iegūtu labākus rezultātus izmantojot trenētu mašīnmācīšanās modeli.

Vārdnīca – Datu struktūra, kas nodrošina piekļuvi datiem, programmām vai datnēm, izmantojot to identifikatorus[10].

IEVADS

Kvalifikācijas darba mērķis ir izstrādāt papildus funkcionalitātes prototipa implementāciju pasūtītāja mašīnmānīšanās teksta analizatorā, tās sniedzot *MMTA* lietotājiem vairāk iespēju izvēlēties no vairākiem tēmu modelēšanas paņēmeniem analizējot lietotāja izvēlētos teksta datus.

Izstrāde ir nepieciešama, jo katra tēmu modelēšanas metode strādā pamotojoties uz dažādiem pieņēmumiem. Dodot lietotājam izvēli starp vairākām metodēm, lietotājs *MMTA* lietošanas laikā var atrast metodi, kas pēc lietotāja domām sniedz labāko rezultātu jeb lietotājs var izvēlēties metodi, kas viņprāt vislabāk raksturo analizējamos datus.

Iepriekšminēto iemeslu dēļ pasūtītājs ir izvēlējies korelāciju ekstrakcijas(*CorEx*[7]) metodi kā nākamo implementējamo metodi savā sistēmā, un šo implementāciju var veikt izmantojot atvērtā primkoda risinājums *corextopic*[3] integrēšanu pasūtītāja sistēmā. *Corextopic*[3] ir atvērtā primkoda risinājums, kas realizē *CorEx*[7] metodes darbību

Integrēšanas veikšanai tiks izmantota *Python 3.5.5* programmēšanas valoda, objektorientētas programmēšanas pieeja un atvērtā pirmkoda risinājumi kā *corextopic*[3], C*ountVectorizer, SnowBallStemmer, WordNetLemmatizer* un citus.

Sākotnējās konsultācijās ar pasūtītāju tika nolemts, ka jātiek izveidotam konceptuālam prototipam, ko lēnām izstrādes laikā papildinot tiks izveidota jau pilnīgi lietojama *CorEx*[7] implemetnācija sistēmā ar tās pamatfunkcijām, kuras ir ielasīt konfigurācijas failu, ielasīt lietotāja dotos ievaddatus un kā rezultātu izvadīt trīs failus – sakategorizētus ievaddatus divos veidos un kategoriju jeb tēmu skaidrojumu.

Izpildot iepriekšējā teikumā minētās prasības, var tālāk tikt veikta *CorEx*[7] implementācijas funkcionāla papildināšana un implementētā risinājuma darbības uzlabošana.

1. PROGRAMMATŪRAS PRASĪBU SPECIFIKĀCIJA

1.1.Ievads

1.1.1. Nolūks

Programmatūras prasību specifikācija (turpmāk PPS) ir dokuments, kas raksturo prasības, kas jāievēro *corextopic*[3] bibliotēkas implementācijas veikšanai. Implementācijas mērķis ir papildināt *MMTA* tēmu modelēšanas arsenālu ar bibliotēku, kas ir spējīga teksta dokumentu datu kopā atrast iedalījumu tēmās un atpazīt kādas tēmas piemīt katram dokumentam dotajā datu kopā.

1.1.2. Darbības sfēra

Izstrādātā sistēma jeb *CorEx*[7] *implementācija* ir lielākas sistēmas (*MMTA*) funkcionalitātes papildinājums, kas nodrošina teksta dokumentu kategorizēšanu izmantojot *corextopic*[3] mašīnmācīšanās bibliotēku.

1.1.3. Saistība ar citiem dokumentiem

Dokumenta noformēšanā ievērotas standarta LVS 68:1996 „Programmatūras prasību specifikācijas ceļvedis” prasības.

1.1.4. Pārskats

PPS sastāv no šādām nodaļām:

1) Ievads. Apraksta PPS nolūku, darbības sfēru, saistību ar citiem dokumentiem.
2) Vispārējais apraksts. Satur vispārēju aprakstu par sistēmas pamatfunkcionalitāti, tās perspektīvu, sistēmas ierobežojumiem un sistēmas lietotāja raksturiezīmes.
3) Konkrētās prasības. Detalizēti specificētas visas prasības, kas sistēmas izstrādātājam jāievēro, lai veiktu sistēmas projektējumu.
4) Ārējā saskarne. Aprakstītas gan lietotāja, gan sistēmas saskarnes
5) Nefunkcionālās prasības. Aprakstītas sistēmas veikstpējas prasības.
6) Konceptuālās datu plūsmas. Attēlota sistēmas darbība izmantojot diagrammas.

1.2. Vispārējais apraksts

1.2.1. Produkta perspektīva

CorEx[7] implementācija ir veikta atbilstoši pasūtītāja prasībām, mērķiem un vajadzībām, ievērojot Latvijas Republikas un Eiropas Savienības likuma prasības. Izstrādātās sistēmas fukcionālo darbību nodrošina *CorEx*[7] implementācijas saskarne ar *MMTA*. Savukārt *MMTA* ir programmprodukts, kuras darbību un lietojamību nodrošina pasūtītājs.

Izstrādātā sistēma izmanto *MMTA* iekšējās funkcijas failu ielasīšanai un rakstīšanai, sistēmas darbības konfigurēšanai, klašu un koda struktūras pārbaudei. Izmantojot *MMTA* iekšējās funkcijas, tiek nodrošināts, ka nākotnē implementēto fukcionalitāšu darbības laikā faili tiks ielasīti un rakstīti vienādā formātā un ka tiks ievērota *MMTA* projekta programmatūras koda struktūra.

1.2.2. Produkta funkcijas

CorEx[7] implementācijas fukcionalitāte ir sistematizēta pēc klasēm un pēc to nozīmes un pamatmērķa teksta analīzes processā. Zemāk ir redzamas klases, kas nodrošina sistēmas un implementētās funckionalitātes kopēju darbību (implementācijas klases, *MMTA* klases un bibliotēkas klases). Implementācijas klases ir klases, kas ir jāizstrādā. *MMTA* klases ir jau sistēmā izstrādātas klases, kurām pēc vajadzības ir jāveic izmaiņas, lai veidotos saskarne ar implementācijas laikā izstrādātajām klasēm. Bibliotēkas klases ir klases, kuru funkcijas tiešā veidā izmanto implementācijas laikā izstrādātajās klases. Pēc pasūtītāja lūguma, pielietotās *MMTA* klases tiks raksturotas vispārēji. Bibliotēkas klases tiks aprakstītas vispārēji, jo to oficiālā dokumentācija ir atrodama atsaucēs.

Implementācijas klases:

- *CorExExtractor*
- *CorExConfiguration*

MMTA klases:

- *TopicModeling*
- *Trainer*
- *FileHelper*

- *BaseExtractor*
- *Factory*
- *RunConfiguration*

Bibliotēkas bibliotēkas klase:

- *CorExModel*[3]

Tālāk seko katras klases darbības apraksts:

TopicModeling: Uzsāk *MMTA* darbību izmantojot *Trainer* klasi.

Trainer: Izmantojot klases *FileHelper* un *Factory,* ielasa konfigurācijas failu, izveido *CorExExtractor* instanci un izsauc *CorExExtractor* vajadzīgās funkcijas vajadzīgajā secībā, lai tiktu veikta teksta analīze.

FileHelper: Definē un veic dažādu formātu failu ielasīšanu un rakstīšanu.

BaseExtractor: Pārbauda vai *CorExExtractor* klase satur projekta darbībai nepieciešamās funkcijas.

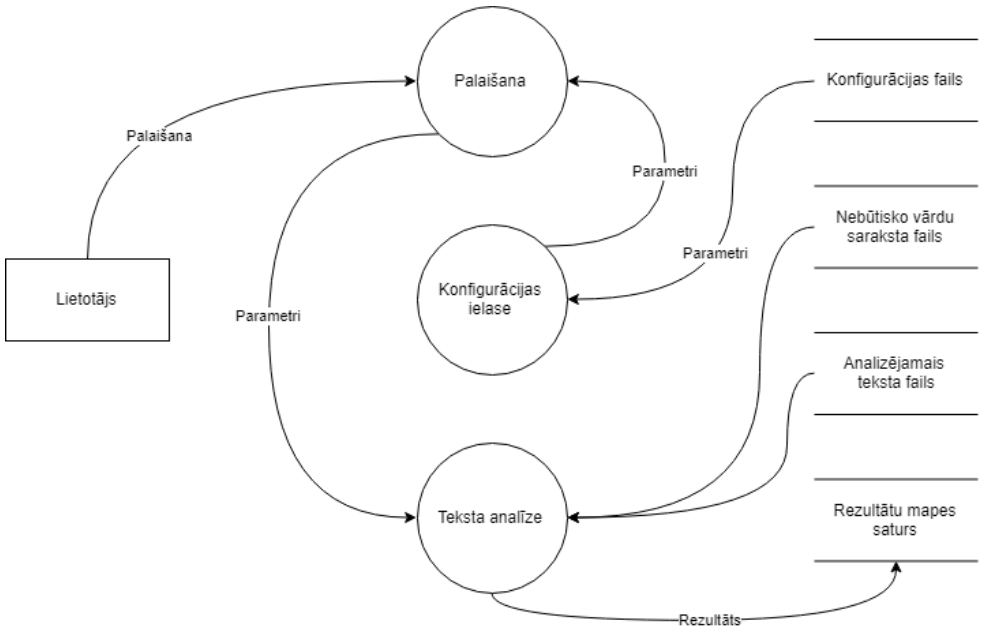
Factory: Izmantojot klases *RunConfiguration* un *CorExConfiguration,* saglabā lietotāja definējamās mainīgās vērtības, kas atrodamas produkta konfigurācijas failā, kā arī klasei *Trainer* dod piekļuvi klases *CorExExtractor* funkcijām.

RunConfiguration: ielasa konfigurācijas faila sadaļu "*RUN"*

CorExConfiguration: ielasa konfigurācijas faila sadaļu "*COREX"*

CorExExtractor: satur vajadzīgās funkcijas, lai nodrošinātu *corextopic*[3] biliotēkas implementāciju *MMTA*

Attēlā (att.) redzams *MMTA* darbības koncepts. *Corextopic*[3] bibliotēkas implementācijai jānotiek *MMTA* darbības sadaļā "Teksta analīze". Lai gan pēc vajadzības tā var arī notikt citās sadaļās



Att 1.2.2.1 ***MMTA*** **darbības koncepts**

1.2.3. Lietotāja raksturiezīmes

MMTA lietotājs ir izstrādātās sistēmas lietotājs. Lietotājs ir tāds, kuram ir pasūtītāja piešķirta license izmantot *MMTA* vai arī pasūtītāja darbinieks, kas izmanto *MMTA* darba vajadzībai. *MMTA*. Sistēmas saskarne tiek realizēta izmantojot.csv un .cfg kā arī vai nu .dockerfile, vai nu .exe, failus, kas nosaka, ka *MMTA* lietotājam ir nepieciešamas labas zināšanas darbā ar datoru kā arī vismaz vidējas zināšanas par mašīnmācīšanos un teksta analīzi.

1.2.4. Vispārēji ierobežojumi

Sistēmas darbība ir iespējama izmantojot Windows un Linux operētājsistēmas. Vai arī sistēmas, kas spēj palaist *windows executables* vai *dockerfile*, jo *MMTA* var tikt kompilēts gan kā *windows executable* fails, gan kā *dockerfile* fails.

1.3.Konkrētās prasības

1.3.1. Vispārīgā informācija

CorEx[7] *implementācijas* funkcionalitātes izklāsts strukturēts skatoties uz katras funkcijas piederību kādam sistēmas konceptuālajam modulim. Sistēmas *Corex*[7] *implementācija* moduļi izšķirami sekojoši:

- Palaišana
  - *TopicModeling* - *MMTA* sākuma punkts
  - *Trainer* - Ekstraktora vadīšana
  - *Factory* - Saskarne ar konfigurācijas ielasītājiem un ekstraktoriem
- Konfigurācijas ielase
  - *RunConfiguration* - Palaišanas konfigurācijas ielasītājs
  - *CorExConfiguration* - Ekstraktora konfigurācijas ielasītājs
- Teksta Analīze
  - *FileHelper* - Failu rakstīšana, lasīšana
  - *CorExExtractor* – Ekstraktors

Pēc pasūtītāja lūguma detalizēti tiks apskatītas tikai *CorEx*[7] *konfigurācijas* un *CorEx*[7] *ekstrakcijas* moduļi, pārējie moduļi tiks aprakstīti, lai būtu saprotama to pamatdarbība.

Prasību specifikācijas trasējamības realizēšanai, katra funkcija tiek identificēta ar unikālu identifikatoru. Identifikatoru veido funkcijas piederība klasei un funkcijas nosaukums rakstīts ar lielajiem burtiem. Klases un funkcijas nosaukums ir atdalāms ar punktu jeb "." simbolu. Ja klases nosaukumu veido vairāki vārdi, tad vārdi tiek rakstīti kopā (bez atstarpes). Katru vārdus klases nosaukumā raksta ar lielo pirmo burtu, bet pārējos burtus ar mazajiem burtiem. Savukārt vārdus funkcijas nosaukumā atdala ar apakšsvītru jeb "_" simbolu. Visi vārdi funkcijas nosaukumā tiek rakstīt ar mazajiem burtiem. Identifikatora beigās tiek pierakstītas atvērošās un aizverošās iekavas jeb "()" simbolu kombinācija. Ja identifikators satur tikai klases nosaukumu, tad tas ir uzskatāms par klases identifikatoru.

Identifikatoru piemēri:

- *CorExExtractor.save_models()*-funkcija;

- *CorExConfiguration.save_as_dictionary()* - funkcija*;*
- *Trainer* - klase*;*
- *FileHelper* - klase*;*
- *FileHelper.read_csv()* - funkcija

Visām sistēmas funkcijām ir sekojošas kopīgas iezīmes :

- Visas sistēmā pielietotās simbolu virknes tiek kodētas *UTF8*.
- Ja nav citādāk minēts, tad ievaddati ir obligāti un citādāk izsauc kļūdas ziņojumu par trūkstošo ievaddatu neesamību.
- *None, False* un *True* nav klašu vai funkciju identifikatori, tās ir mainīgo vērtības

1.3.2. Funkcionālās prasības

1.3.2.1. Palaišana

1.3.2.1.1. TopicModeling

1.1 tabula

Tēmu modelēšanas progammas sākumpunkts

Identifikators: TopicModeling
Mērķis
Uzsāk *MMTA* darbību izmantojot *Trainer* klasi.

1.2 tabula

Tēmu modelēšanas palaišanas funkcija

Identifikators: TopicModeling.main()
Mērķis
Izveido *Trainer* klases instanci izmantojot *Trainer.init()* un izsauc *Trainer.start_services()*
Ievaddati
• Konfigurācijas fails
Izvaddati
• Konsolē tiek izvadīts paziņojums “Tēmu Modelēšana ir sākta"

1.3.2.1.2. Trainer

1.3 tabula

Tēmu modelēšanas klase Trainer

Identifikators: Trainer
Mērķis
Izveidot saskarni starp *TopicModeling* klasi un lietotāja izvēlēto teksta analīzes metodi jeb ekstraktoru. Sistēmas izstrādāšanas laikā tiek apskatīts tikai *CorExExtractor* ekstraktors.

1.4 tabula

Trainer klases inicializācija

Identifikators: Trainer.init()
Mērķis
Inicializē *Factory* klases instanci, izsauc funkcijas *Factory.create_run_configuration()* un *Factory. create_extractor_configuration()*
Ievaddati
• Konfigurācijas fails

1.5 tabula

Trainer klases darbības sākumpunkts

Identifikators: Trainer. start_services()
Mērķis
Izsauc *Trainer.run_training()*
Ievaddati
• Konfigurācijas fails

1.6 tabula

Ekstraktoru izveides, vadības un konfigurācijas failu ielases sākumpunkts

Identifikators: Trainer.run_training()
Mērķis

Izveido *Factory* klases instanci izmantojot funkciju *Factory.init()* un izsauc *Factory.create_extractor(), CorExExtractor.train(), CorExExctractor.extract()* un *CorExExtractor.save_models(),*
Ievaddati
• Konfigurācijas fails

1.3.2.1.3. Factory

1.7 tabula

Factory

Identifikators: Factory
Mērķis
Veido saskarni ar konfigurācijas faila sadaļu lasītājiem (piemēram *RunConfiguration*) un ekstraktoriem (piemēram *CorExExtractor*). Izmantojot klases *RunConfiguration* un *CorExConfiguration,* saglabā lietotāja definējamās mainīgās vērtības, kas atrodamas *MMTA* konfigurācijas failā, klasei *Trainer* dod piekļuvi klašu *CorExExtractor, RunConfiguration, CorExConfiguration* funkcijām.

1.7 tabula

Inicializācija

Identifikators: Factory.init()
Mērķis
Izveidot klases Factory instanci
Ievaddati
• Nav
Apstrāde
1. Izveido sarakstu, kas satur informāciju par to kādi faili ir vajadzīgi, lai izveidotu konfigurācijas failā izvēlēto ekstraktoru un ielasītu izvēlētā ekstraktora konfigurācijas parametrus no konfigurācijas faila.
Izvaddati
• Ekstraktora parametri (*CorExConfiguration.init()*)
Kļūdu paziņojumi

• nav

1.8 tabula

Palaišanas konfigurācijas ielase

Identifikators: Factory.create_run_configuration()
Mērķis
Izveidot klases *RunConfiguration* instanci izsaucot *RunConfiguration.init()*
Ievaddati
• Konfigurācijas fails
Apstrāde
1. Nav
Izvaddati
• Palaišanas parametri (RUN konfigurācijas sadaļa)
Kļūdu paziņojumi
• Nav

1.9 tabula

Ekstraktora konfigurācijas ielase

Identifikators: Factory.create_extractor_configuration()
Mērķis
Izveidot ekstraktora klases (*CorExExtractor*) instanci, ko lietotāja uzstādījis RUN konfigurācijas sadaļā
Ievaddati
• Konfigurācijas fails • Ekstraktora tips (“CorEx*”*)
Apstrāde
2. Apskatot
Izvaddati
• Ekstraktora parametri (*CorExConfiguration.init()*)
Kļūdu paziņojumi
•

1.10 tabula

Ekstraktora klases izveide

Identifikators: Factory.create_extractor()
Mērķis
Izsauc CorExExtractor.init()
Ievaddati
• Ekstraktora tips (šajā dokumentā simbolu virkne "CorEx") • Ekstraktora parametri (*CorExConfiguration.init()*)
Apstrāde
1.
Izvaddati
•
Kļūdu paziņojumi
•

1.3.2.2. Konfigurācijas ielase

1.3.2.2.1. RunConfiguration

1.11 tabula

RunConfiguration

Identifikators: RunConfiguration
Mērķis
Ielasīt lietotāja noteiktās vērtības konfigurāciajs failā un darbināt *MMTA* izmantojot šīs vērtības

1.12 tabula

Inicializācija. Palaišanas konfigurācijas ielase

Identifikators: RunConfiguration.init()
Mērķis
Inicializē *RunConfiguration* klases instanci, ielasot, pārbaudot un saglabājot konfigurācijas faila sadaļu RUN

Ievaddati
• Konfigurācijas fails
Izvaddati
• Nav
Apstrāde
1. Pārbauda, vai konfigurācijas parametru vērtibas atbilst pareizajiem datu tipiem
Izvaddati
•
Kļūdu paziņojumi
•

1.3.2.2.2. CorExConfiguration

1.13 tabula

CorExConfiguration

Identifikators: CorExConfiguration
Mērķis

1.14 tabula

Inicializācija. Ekstraktora konfigurācijas ielase

Identifikators: CorExConfiguration.init()
Mērķis
Ievaddati
•
Apstrāde
1.
Izvaddati
•
Kļūdu paziņojumi

•

1.3.2.3. Teksta analīze

1.3.2.3.1. CorExExtractor

1.15 tabula

CorExExtracotr

Identifikators: CorExExtractor
Mērķis
Veikt *corextopic*[3] bibliotēkas implementēšanu *MMTA*, sekojot projekta koda struktūrai pēc pasūtītāja prasībām.

1.16 tabula

Inicializācija

Identifikators: CorExExtractor.init()
Mērķis
Ievaddati
•
Apstrāde
1.
Izvaddati
•
Kļūdu paziņojumi
•

1.17 tabula

Ekstraktora procesu organizēšana

Identifikators: CorExExtractor.train()
Mērķis

Izsaukt vairākas *CorExExtractor* klases funkcijas, lai, pirmkārt, pārbaudītu *CorExExtractor* parametru pareizību (*CorExExtractor.check_valid_config()*); otrkārt, sagatovotu vai nu vārdu lemmatizāciju, vai nu vārdu sakņu atdalīšanu no vārdiem (*CorExExtractor.load_stemmer_or_lemmatizer()*); treškārt ielasīt nebūtisko vārdu sarakstu (*CorExExtractor.load_stopwords()*); ceturtkārt, veikt apstrādājamā teksta pirmsapstrādi (*CorExExtractor.preprocess_text()*); piektkārt, aprēķināt vārdu (terminu jeb angļu valodā *term*) biežumu (*CorExExtractor.calculate_term_frequency()*); sestkārt, izveidotu *corextopic*[3] bibliotēkā pieejamo mašīnmācīšanās modeli(*CorExExtractor.train_with_corex()*).
Ievaddati
• Ievaddatu mapes atrašanās vieta. Simbolu virkne; • Ievaddatu faila nosaukums. Simbolu virkne; • Ievaddatu atdalošais simbols. Simbolu virkne; • Ievaddatu galvenes esamība. Būla tips; • Ievaddatu tekstu saturošās kolonnas numurs. Vesela skaitliska vērtība.
Apstrāde
1. Izsauc funkciju *CorExExtractor.check_valid_config()* 2. Izsauc funkciju *CorExExtractor.load_stemmer_or_lemmatizer()*; 3. Izsauc funkciju *CorExExtractor.load_stopwords()*; 4. Izsauc funkciju *CorExExtractor.preprocess_text();* 5. Izsauc funkciju *CorExExtractor.calculate_term_frequency();* 6. Izsauc funkciju *CorExExtractor.train_with_corex()*.
Izvaddati
• Nav
Kļūdu paziņojumi
• "Ievaddatu mape X neeksistē", kur X vietā ievieto lietotāja specificētu Ievaddatu mapes atrašanās vietu, ja šāda mape nav atrodama.

1.18 tabula

Rezultātu ekstrakcija

Identifikators: CorExExctractor.extract()
Mērķis

Pēc funkcijas *CorExExtractor.train()* darbības apstrādāt datus izmantojot *CorExModel.transform()*[3], lai funkcija *CorExExtractor.save_models()* varētu saglabāt rezultātus failos pareizā formātā.

Ievaddati

- Nav

Apstrāde

1. Izsauc funkciju *CorExModel.transform()*[3]
2. Saglabā *CorExModel.transform()*[3] izvadīto matricu *p_y_given_x*[3] *(*saīsinājums no: *probability of Y for a given X* [7], nozīme:satur matricu, kur kolonnas atbilst tēmām, rindas atbilst teksta dokumentiem un vērtības ir decimāldaļskaitļi lielumā 0 – 1, kas apzīmē izveidotā modeļa pārliecību par katra dotā dokumenta piederību katrai dotai tēmai, kur 0 nozīmē pilnīgu nepiederību un 1 nozīmē pilnīgu piederību konkrētai tēmai*)* mainīgajā *docTopicDist;*
3. Iterējot cauri *docTopicDist* sastāda mainīgos:
    - *labeledDocs* – matrica ar trim kolonām. Pirmā kolonna satur dokumentu tekstus, otrā - tēmas numurs, ar lielāko pārliecību dotajam dokumentam, trešā – konkrētās tēmas pārliecības lielums;
    - *multilabelData* – matrica ar neregulāru kolonnu skaitu katrā rindā. Katrai rindai ir vismaz viena kolonna. Pirmā kolonna – satur dokumentu tekstus. Otrā kolonna – tēmas numurs, ar lielāko pārliecību dotajam dokumentam, kas pārsniedz lietotāja uzstādīto *ProbabilityThreshold* (pārliecības sliekšņvērtība) konfigurācijas faila sadaļā COREX
    - *docTopicScores* – vārdnīca (*dictionary*), kuras atslēga (*key*) ir katra dokumenta kārtas numurs, un katras atslēgas vērtība (*value*) ir saraksts (*array*) ar apakšsarakstiem (*nested array*), kuri satur divas vērtības: pirmā - tēmas numurs, otrā – modeļa pārliecība par doto tēmu dotajam dokumentam. Katra saraksta apakšsaraksti ir izkārtoti secībā pēc katra apakšaraksta otrās vērtības (pārliecības) dilstošā secībā.

Izvaddati

- Konsolē izvada paziņojumu “Notiek rezultātu apstrāde”

Kļūdu paziņojumi

•

1.19 tabula

Rezultātu saglabāšana

Identifikators: CorExExtractor.save_models()
Mērķis
Izmantojot *FileHelper* klases funkcijas saglabāt ar funkciju *CorExExtractor.extract(),* izveidotos un aizpildītos mainīgos.
Ievaddati
• Izvaddatu (rezultātu) mapes atrašanās vieta. Simbolu virkne
Apstrāde
1. Saglabā mainīgo *labeledDocs* failā ar nosaukumu *labeledDocuments.csv* izmantojot *FileHelper.save_to_csv()* 2. Saglabā mainīgo *multilabelData* failā ar nosaukumu *multilabelData.csv* izmantojot *FileHelper.save_to_csv()* 3. Saglabā mainīgo *docTopicScores* failā ar nosaukumu *topicScores.json* izmantojot *FileHelper.save_json()* 4. Saglabā mainīgo *topics* failā ar nosaukumu *topicsByKeywords.csv* izmantojot *FileHelper.save_to_csv()*
Izvaddati
•
Kļūdu paziņojumi
•

1.20 tabula

Konfigurācijas vērtību pārbaude

Identifikators: CorExExtractor.check_config_valid()
Mērķis
Pārbaudīt konfigurācijas failā atrodamos COREX parametrus un to datu tipus.
Ievaddati
• No konfigurācijas faila ielasīto parametru vārdīca.

• *CorExExtractor.init()* izveidota un cieti aizpildīta vārdnīca, kas satur informāciju par to, kāda tipa katram parametram jābūt, un kāda ir parametra noklusējuma vērtība.
Apstrāde
1. Salīdzina pirmās vārdnīcas vērtibu datu tipus ar otrajā vārdnīcā definētajām vērtībam salīdznot vērtības, kurām ir vienādas atslēgas. 2. Ja kāda atslēga ir otrajā vārdnīcā, bet ne pirmajā, tad izvada kļūdas paziņojumu 1 3. Ja kādas pirmās vārdnīcas tips nav atbilstoš, tad izvada kļūdas paziņojumu 2 4. Ja kādas pirmās vārdnīcas vērtiba ir *None*, tad piešķir otrajā vārdnīcā definēto parametra noklusējuma vērtību
Izvaddati
• Ja ir izpildījies 4. apstrādes soļa nosacījums, tad konsolē izvada ziņojumu “Parametram X ir piešķirta noklusējuma vērtība Y”, kur X tiek aizvietots ar parametra nosaukumu, pie kura ir izpildījies 4. apstrādes soļa nosacījums un Y ir šī parametra noklusējuma vērtība.
Kļūdu paziņojumi
1. "Parametru X nevar atrast konfigurācijā", kur X tiek aizvietots ar trūkstošo parametra nosaukumu; 2. “Parametram X jābūt ir vai nu tipa Y, vai nu vērtībai None”, kur X ir tiek aizvietots ar neatbilstošā tipa parametra nosaukumu un Y tiek aizvietots ar pareizo parametra tipa nosaukumu.

1.21 tabula

Pirmsapstrādes ielāde

Identifikators: CorExExtractor.load_stemmer_or_lemmatizer()
Mērķis
Pēc konfigurācijas failā zem sadaļas COREX lietotāja izvēlētā teksta pirmsapstrādes metodes ielādēt un izveidot izvēlētās teksta pirmsapstrādes metodes klases instanci.
Ievaddati
• Konfigurācijas parametrs *applyStemmingOrLemmatization*
Apstrāde

1. Ja *applyStemmingOrLemmatization* vērtība ir simbolu virkne “stemming” izveidot *snowballStemmer* klasi, kas ir atvērtā pirmkoda bibliotēka. Ja rodas problēmas ielādējot bibliotēku, izvada kļūdu paziņojumu 1. 2. Ja *applyStemmingOrLemmatization* vērtība ir simbolu virkne “lemmatization” un izveidot *wordNetLemmatizer* klasi, kas ir atvērtā pirmkoda bibliotēka. Ja rodas problēmas ielādējot bibliotēku, izvada kļūdu paziņojumu 1. 3. Ja *applyStemmingOrLemmatization* vērtība ir *None* tad turpināt darbu.
Izvaddati
• Ja izpildījies 3. apstrādes soļa nosacījums, tad izvadīt konsolē ziņojumu “Vārdu celmošana un lemmatizēšana nenotiks, jo applyStemmingOrLemmatization vērtība ir None”
Kļūdu paziņojumi
1. “Nav izdevies ielādēt X bibliotēku. Pārbaudiet ieinstalētās bibliotēkas”, kur X ir trūkstošā bibliotēka.

1.22 tabula

Nebūtisko vārdu ielasīšana

Identifikators: CorExExtractor.load_stopwords()
Mērķis
Izņemt nebūtiskos vārdus no teksta dokumentiem teksta dokumentu datu kopā, ja , konfigurācijas failā zem sadaļas COREX lietotājs ir definējis, *.csv* faila nosaukumu, kas satur nebūtisko vārdu sarakstu.
Ievaddati
• Ievaddatu mapes atrašanās vieta. Simbolu virkne; • Konfigurācijas parametrs *removeStopwords.* Būla tips • Konfigurācijas parametrs *stopwordsFileName.* Vai nu *None,* vai nu simbolu virkne.
Apstrāde
1. Ja *removeStopwords* vērtība ir *True*, tad veikt apstrādes 2. soli 2. Ja *stopwordsFileName* vērtība ir *None* vai tukša simbolu virkne, tad no atvērtā pirmkoda bibliotēkas *nltk.corpus* ielādēt *stopwords* un saglabāt mainīgajā *stopWordList* kā *set* tipa sarakstu. Ja rodas problēmas ielādējot bibliotēku, izvada kļūdu paziņojumu 1.

3. Ja neizpildās 2. apstrādes soļa nosacījums, tad ielādēt no ievaddatu mapes failu, kura nosaukums ir vienāds ar mainīgā *stopwordsFileName* vērtību. Ja fails nav atrodams, tad izvadīt kļūdu paziņojumu 2.
Izvaddati
•
Kļūdu paziņojumi
1. "Nav izdevies ielādēt X bibliotēku. Pārbaudiet ieinstalētās bibliotēkas", kur X ir trūkstošā bibliotēka. 2. "Nav izdevies atrast failu X mapē Y.", kur X ir *stopwordsFileName* vērtība, bet Y ir ievaddatu mapes vērtība.

1.23 tabula

Teksta pirmsapstrāde

Identifikators: CorExExtractor.preprocess_text()
Mērķis
Tokenizēt ielādēto teksta dokumentu saturu
Ievaddati
• Teksta dokumentus saturošā faila atrašānās vieta mainīgajā *textData*. Simbolu virkne; • Ievaddatu atdalošais simbols mainīgajā *dataDelimiter*. Simbolu virkne; • Ievaddatu galvenes esamība mainīgajā *dataFileHasHeader*. Būla tips; • Teksta dokumentu datu kopā tekstu saturošās kolonnas numurs mainīgajā *dataTextColumn*. Vesela skaitliska vērtība.
Apstrāde
1. Ja *textData* fails nav atrodams izvadīt kļūdu ziņojumu 1.; 2. Izveidot klases *FileHelper* instanci un mainīgajā *documents* ielasīt teksta dokumentus izmantojot *FileHelper.read_csv()*; 3. Iterējot cauri mainīgā *documents* rindām, katrā iterācijā izmantojot funckiju *CorExExtractor.turn_text_into_tokens()* tokenizēt *documents* rindu un izveodotos tokenus apvienot starp tiem ievietojot atstarpi un pievienot mainīgajam *documentsProcessed*, kas ir saraksts
Izvaddati

• *documentsProcessed.* Saraksts ar apstrādātiem teksta dokumentiem
Kļūdu paziņojumi
1. "Nav idevies atrast failu X", kur X ir teksta dokumentus saturošā faila atrašānās vietas vērtība

1.24 tabula

Teksta vektorizēšana

Identifikators: CorExExtractor.calculate_term_frequency()
Mērķis
Aprēķināt terminu sastopamību un saglabāt mainīgajā *termFrequency*, kā arī izveidot un aizpildīt mainīgo *featureNames*, kam jāsatur teksta dokumentu datu kopas unikālo vārdu sarakstu alfabētiskā secība. Izmantojot atvērtā pirmkoda (*open-source*) bibliotēku *countVectorizer* pārveidot teksta dokumentu datu kopu par matricu, kuras kolonnas ir unikālo vārdu alfabētiska secība, un rindas ir teksta dokumentu secība, un vērtibas attēlo to, cik reizes dotais vārds ir atrodams dotajā dokumentā.
Ievaddati
• Apstrādāta teksta dokumentu datu kopa mainīgajā *listDocuments*
Apstrāde
1. Izmantojot *countVectorizer* funkciju *fit_transform* uz mainīgā *listDocuments* aizpildīt mainīgo *termFrequency;* 2. Izmantojot bibliotēkas *scipy.sparse* funkciju *csr_matrix* apstrādāt *termFrequency;* 3. Izmantojot *countVectorizer* funkciju *get_feature_names* aizpildīt mainīgo *featureNames;*
Izvaddati
• *termFrequency.* Matrica; • *featureNames. S*imbolu virķnu saraksts.
Kļūdu paziņojumi
• Nav

1.25 tabula

Corex modeļa trenēšana

Identifikators: CorExExtractor.train_with_corex()
Mērķis
Izveidot *corextopic*[3] bibliotēkā pieejamo mašīnmācīšanās modeli un saglabāt to kā klasi ar nosaukumu *CorExModel* izsaucot funkciju *CorExModel.corex()*[3]
Ievaddati
• Ievaddatu faila nosaukums. Simbolu virkne; • ar *CorExExtractor.calculate_term_frequency()* izveidotā datne saglabāta mainīgajā *termFrequency.*; • ar *CorExExtractor.calculate_term_frequency()* izveidotā datne saglabāta mainīgajā *featureNames.* Neobligāta vērtība. Pēc noklusējuma - *None*
Apstrāde
1. Izveido *CorExModel* klasi (mašīnmācīšnās modeli) izmantojot *corextopic* bibliotēkas iekšējo funkciju *corex()[3]* (dokumentā līdzvērtīga funkcijai ar identifikatoru *CorExModel.corex()*[3]) 2. Izmantojot funkciju *CorExModel.fit()*[3] veic mašīnmācīšnās modeļa pielāgošanu (*fitting*) teksta dokumentu datu kopai
Izvaddati
• Konsolē izvada “Sākusies CorEx inicializācija un trenēšana”
Kļūdu paziņojumi
• Nav

1.3.2.3.2. CorExModel

1.26 tabula

CorExModel

Identifikators: CorExModel
Mērķis
Izveidot *corextopic*[3] biliotēkā pieejamo mašīnmācīšanās modeļa, klases instanci un izveidot pieeju šīs bibliotēkas funkcijām. Saglabāt instanci ar nosaukumu *CorExModel*

1.27 tabula

Inicializājica

Identifikators: CorExModel.corex()
Mērķis
Mašīnmācīšanās modeļa izvede izmantojot *corextopic*[3] bibliotēku.
Ievaddati
• Konfigurācijas parametrs *numOfTopics,* • Konfigurācijas parametrs *numOfIter,* • Konfigurācijas parametrs *randomSeed* • Funckijā *CorExExtractor.calculate_term_frequency()* izveidotais un aizpildītais mainīgais *featureNames*
Apstrāde
1. Saktīt atsauci par *corextopic*[3]
Izvaddati
• Inicializēts mašīnmācīšanās modelis • Saktīt atsauci par *corextopic*[3]
Kļūdu paziņojumi
• Saktīt atsauci par *corextopic*[3]

Modeļa pielāgošana

Identifikators: CorExModel.fit()
Mērķis

Pielāgot (trenēt) mašīnmācīšanās modeli padotajiem datiem. Vairāk par funkciju lasāms: skatot atsauci par *corextopic*[3]
Ievaddati
• Konfigurācijas COREX sadaļas parametrs *numOfTopics;* • Konfigurācijas COREX sadaļas parametrs *numOfIter;* • Konfigurācijas DEVELOPER sadaļas parametrs *randomSeed;* • Funkcijā *CorExExtractor.calculate_term_frequency()* iegūtais un aizpildītais mainīgais *featureNames*
Apstrāde
1. Saktīt atsauci par *corextopic*[3]
Izvaddati
• Trenēts mašīnmācīšanās modelis • Saktīt atsauci par *corextopic*[3]
Kļūdu paziņojumi
• Saktīt atsauci par *corextopic*[3]

1.28 tabula

Rezultātu iegūšana

Identifikators: CorExModel.transform()
Mērķis
Transformēt padotos datus izmantojot trenēto mašīnmācīšanās modeli atpakaļ saņemot *p_y_given_x* (skatīt funkciju *CorExExctractor.extract()*) Vairāk par *CorExModel.transform()* funkciju lasāms skatot atsauci par *corextopic*[3].
Ievaddati
• Mainīgais *termFrequency*
Apstrāde
1. Saktīt atsauci par *corextopic*[3]
Izvaddati
• Mainīgais *p_y_given_x* • Saktīt atsauci par *corextopic*[3]
Kļūdu paziņojumi

• Saktīt atsauci par *corextopic*[3]

1.3.2.3.3. FileHelper

1.29 tabula

FileHelper

Identifikators: FileHelper
Mērķis
Definē un veic standartizētu dažādu formātu failu ielasīšanu un rakstīšanu

1.30 tabula

Datu kopas rakstīšana .csv failā

Identifikators: FileHelper.save_to_csv()
Mērķis
Saglabā datu kopu *.csv* formāta failā
Ievaddati
• Izvadfaila nosaukums. Simbolu virkne • Datu kopa. Saraksts. • Atdalāmais simbols. Simbols
Apstrāde
1. Izmantojot *Python* iebūvēto funkciju *csv* datu kopu ieraksta *.csv* formāta failā
Izvaddati
• Fails *.csv* formātā, kas satur datu kopas datus.
Kļūdu paziņojumi
• "Faila nosaukuma paplašinājums nav pareizs, fails tiek saglabāts ar paplašinājumu *.csv*"

Datu kopas rakstīšana .pkl failā

Identifikators: FileHelper.save_pickle()
Mērķis
Saglabā datu kopu *.pkl* formāta failā
Ievaddati
• Izvadfaila nosaukums. Simbolu virkne

• Datu kopa. Saraksts. • Atdalāmais simbols. Simbols
Apstrāde
1. Izmantojot *python* iebūvēto funkciju *pickle* datu kopu ieraksta *.pkl* formāta failā
Izvaddati
• Fails *.pkl* formātā, kas satur datu kopas datus.
Kļūdu paziņojumi
• "Faila nosaukuma paplašinājums nav pareizs, fails tiek saglabāts ar paplašinājumu .pkl"

1.31 tabula

Datu kopas rakstīšana .json failā

Identifikators: FileHelper.save_json()
Mērķis
Saglabā datu kopu *.json* formāta failā
Ievaddati
• Izvadfaila nosaukums. Simbolu virkne • Datu kopa. Saraksts. • Atdalāmais simbols. Simbols
Apstrāde
1. Izmantojot *python* iebūvēto funkciju *json* datu kopu ieraksta *.json* formāta failā
Izvaddati
• Fails *.json* formātā, kas satur datu kopas datus.
Kļūdu paziņojumi
• "Datu formāts neatbilst .json formātam" • Faila nosaukuma paplašinājums nav pareizs, fails tiek saglabāts ar paplašinājumu .json"

1.32 tabula

Datu kopas lasīšana no .csv faila

Identifikators: FileHelper.read_csv()
Mērķis
Ielasīt .csv formāta failu datu kopas mainīgajā

Ievaddati
• Ievadfaila nosaukums. Simbolu virkne • Atdalāmais simbols. Simbols.
Apstrāde
2. Ielasa *.csv* formāta failu mainīgajā *fileContents*
Izvaddati
• Mainīgais *fileContents.* Saraksts.
Kļūdu paziņojumi
• "Faila X šifrējums nav lasāms", kur X ir ievadfaila nosaukums • "Fails X nav atrodams", kur X ir ievadfaila nosaukums

1.3.2.3.4. Base Extractor

1.33 tabula

BaseExtractor

Identifikators: BaseExtractor
Mērķis
Pārbaudīt, vai klase *CorExExtractor* satur obligāti vajadzīgās funkcijas. Sekojošām funkcijām ir obligāti jābūt *CorExExtractor* klasē: • *init()* identifikators: *CorExExtractor.init()* • *load_models()* identifikators: *CorExExtractor.load_models()* • *save_models()* identifikators: *CorExExtractor.save_models()* • *train()* identifikators: *CorExExtractor.train()* • *extract()* identifikators: *CorExExtractor.extract()*

1.4.Ārējā saskarne

1.4.1. Lietotāju saskarne

Lietotāja saskarne ar *MMTA* tiek realizēta izmantojot teksta redaktorus un ar *Windows executable* (*.exe*) faila palaišanu. Saskarne tiek organizēta sekojoši:

- *MMTA* mape – satur visus vajadzīgos failus, lai varētu sākt veikt teksta analīzi.

- Fails *configuration.cfg* (konfigurācijas fails) - Konfigurācijas failā lietotājs uzstāda izvēlētos parametrus:
    - Ievaddatu mapes atrašanās vieta;
    - Ievaddatu teksta dokumentus saturošā faila nosaukums;
    - Ievaddatu atdalošais simbols;
    - Ievaddatu galvenes esamība;
    - Ievaddatu tekstu saturošās kolonnas numurs;
    - Rezultātu mapes atrašanās vieta;
    - Tēmu skaits;
    - Iterāciju skaits;
    - Atslēgvārdu skaits;
    - Pārliecības sliekšņvērtība;
    - Unikālo vārdu daudzuma sliekšņvērtība;
    - Izvēle starp vārdu celmošanu, lemamatizēšanu vai neapstrādāšanu;
    - Izvēle starp nebūtisku vārdu noņemšanu vai atstāšanu;
    - Lietotāja izvēlēta nebūtisko vārdu saraksta ielāde vai nebūtisko vārdu saraksta ielāde no bibliotēkas bibliotēkas *nltk.corpus* ar nosaukumu *stopwords;*
- Ievaddatu mape – Satur failus, ko lietotājs vēlās izmantot kā *MMTA* ievaddatus. Šie faili iekļauj sevī sekojošus failus:
    - Ievaddatu teksta dokumentus saturošs fails. Obligāts
    - Nebūtisko vārdu sarakstu saturošs fails. Neobligāts
- *Windows executable* fails – Palaiž *MMTA* ar konfigurācijas failā esošajiem parametriem, apstrādājot ievaddatu teksta dokumentus saturošo failu un teksta analīzes rezultātus saglabājot rezultātu mapē.
- Rezultātu mape – satur *MMTA* iegūtos rezultātus, kas sastāv no šādiem failiem: *labeledDocuments.csv, multilabelData.csv, topicScores.json, topicsByKeywords.csv.*.

1.4.2. Aparatūras saskarne

Izstrādātā sistēma darbojas uz jebkuras sistēmas, kas ir spējīga palaist *.exe* failus un atbalsta *.csv, .json, .cfg* failus. Lietotājam būtu ieteicams izmantot sistēmu ar pilnekrāna, darbvirsmas atbalstu.

1.4.3. Programmatūras saskarne

MMTA ir izveidots izmantojot *Python 3.5.5*. Lai gan *CorEx*[7] *implementācija* pati par sevi iekļauj visas tai vajadzīgās bibliotēkas šobrīd esošajā versijā, tomēr produkta attīstības laikā ir jāseko līdzi šo bibliotēku attīstibai un izmaiņām. Pasūtītāja lūguma dēļ tiks pieminētas dokumentā jau minētās bibliotēkas un dažas populārākās bibliotēkas mašīnmācīšanās nozarē, kas tiek izmantotas *CorEx*[7] *implementācijas ietvaros*. *MMTA* izstrādē jālietoto šādas bibliotēkas:

- *Python* 3.5.5 iekšējās bibliotēkas:
    - *logging* https://docs.python.org/3/library/logging.html
    - *configparser* https://docs.python.org/3/library/configparser.html
    - *csv* https://docs.python.org/3/library/csv.html
    - *json* https://docs.python.org/3/library/json.html
    - *pickle* https://docs.python.org/3/library/pickle.html
    - *os* https://docs.python.org/3/library/os.html
    - *sys* https://docs.python.org/3/library/sys.html
    - *abc* https://docs.python.org/3/library/abc.html
    - *string* https://docs.python.org/3/library/string.html
- Ārējās *Python 3.5.5* bibliotēkas:
    - *Numpy 1.16.0* https://www.numpy.org/
    - *nltk 3.4* https://www.nltk.org/https://www.nltk.org/api/nltk.tokenize.html
    - *corextopic 1.0.2* https://github.com/gregversteeg/corex_topic
    - *sklearn 0.20.2* https://scikit-learn.org/stable/
    - *scipy 1.2.0* https://www.scipy.org/
    - *pandas 0.23.4* https://pandas.pydata.org/

1.4.4. Sakaru saskarne

Izstrādātās sistēmas darbības laikā netiek izmantoti sakari ar citām sistēmām, taču nākotnē sistēmas attīstība varētu iekļaut implementētu komunikāciju ar mājaslapas saskarni izmantojot *API* un *HTTP* vai *HTTPS* protokolus

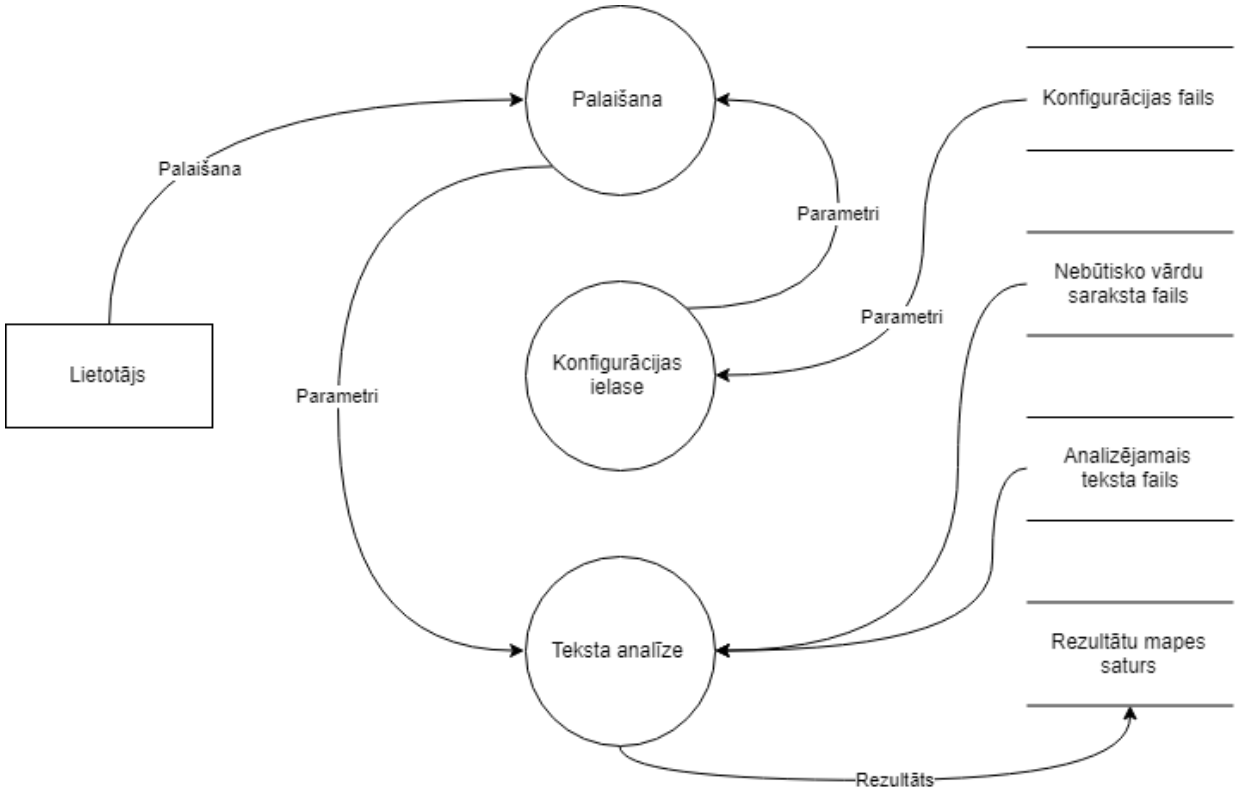
1.5.Nefunkcionālās prasības

1.5.1. Veikstpējas prasības

Apstrādāt 25’000 teksta dokumentu ar 1’000’000 zīmēm 60 sekunžu laikā

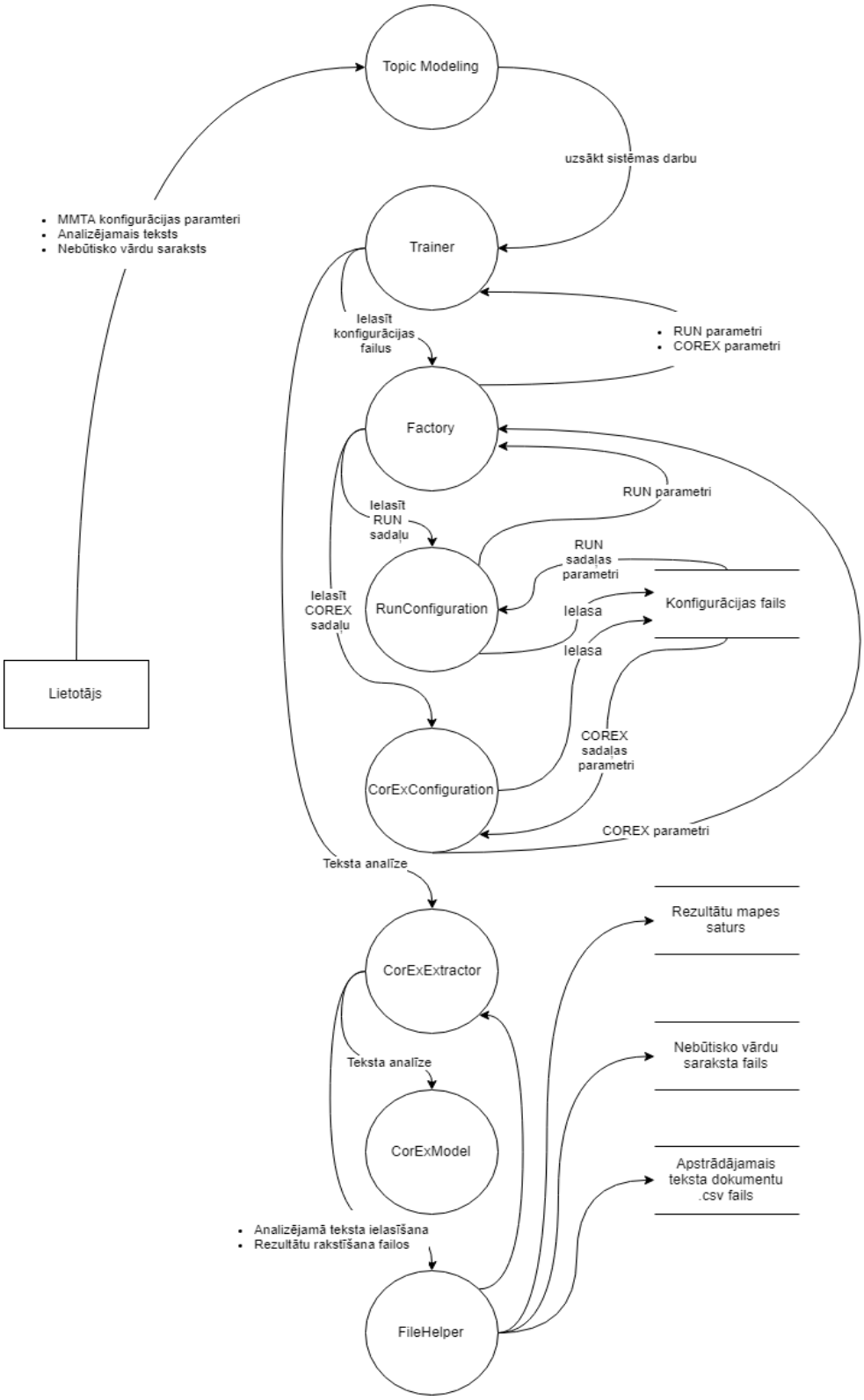
1.6. Konceptuālās datu plūsmas

1.6.1. Sistēmas koncepcija



Att 1.6.1.1 ***CorEx*** **implementāciajs sistēmas koncepcija no lietotāja skatapunkta**

1.6.2. Pirmā līmeņa Datu plūsmu diagramma –



Att 1.6.2.1 ***CorEx*** **implementāciajs sistēmas koncepcija no klašu, moduļu skatapunkta**

2. PROGRAMMATŪRAS PROJEKTĒJUMA APRAKSTS

2.1.Ievads

2.1.1. Nolūks

Programmatūras projektējuma apraksts (turpmāk PPA) apraksta kā tehniski tiks izpildīta *corextopic*[3] bibliotēkas implementācija mašīnmācīšanās teksta analizatorā (turpmāk *MMTA*) pēc dokumentā "Programmatūras prasību specifikācija" (turpmāk PPS) uzskaitītajām prasībām. PPA ir paredzēts *MMTA* izstrādātājiem, *corextopic*[3] bibliotēkas implementācijas labojumiem, izprašanai, *MMTA* izstrādātāju iekšējai lietotšanai.

2.1.2. Darbības sfēra

Aprakstīts PPS nodaļā "Darbības sfēra".

2.1.3. Saistība ar citiem dokumentiem

PPA ir lietojams kopā ar PPS

PPA saturs balstīts uz Valsts standarta : LVS 72:1996 "Ieteicamā prakse programmatūras projektējuma aprakstīšanai" prasībām[12].

2.1.4. Pārskats

PPA sastāv no piecām nodaļām:

Ievads – vispārīga informācija par PPA dokumenta saturu

Tehniskie risinājumi – informācija par *corextopic*[3] bibliotēkas implementāciju

Dekompozīcijas apraksts – aprakstītas projektējuma entītijas

Klašu atkarības - aprakstīts kā realizētas implementācijas klašu attiecības

2.2.Tehniskie risinājumi

Sistēma ir izstrādāta izmantojot *Python 3.5.5,* kā arī sekojošās *Python* iekšējās un ārējās bibliotēkas. Ārējā bibliotēkām ir pierakstīts izmantotās bibliotēkas versija. Abu tipu bibliotēkām ir arī minēta tīmekļa vietne, kurā atrodama konrētās bibliotēkas dokumentācija.

- *Python* 3.5.5 iekšējās bibliotēkas:
    - *logging* https://docs.python.org/3/library/logging.html
    - *configparser* https://docs.python.org/3/library/configparser.html
    - *csv* https://docs.python.org/3/library/csv.html
    - *json* https://docs.python.org/3/library/json.html
    - *pickle* https://docs.python.org/3/library/pickle.html
    - *os* https://docs.python.org/3/library/os.html
    - *sys* https://docs.python.org/3/library/sys.html
    - *abc* https://docs.python.org/3/library/abc.html
    - *string* https://docs.python.org/3/library/string.html
- Ārējās *Python 3.5.5* bibliotēkas:
    - *Numpy 1.16.0* https://www.numpy.org/
    - *nltk 3.4* https://www.nltk.org/https://www.nltk.org/api/nltk.tokenize.html
    - *corextopic 1.0.2* https://github.com/gregversteeg/corex_topic
    - *sklearn 0.20.2* https://scikit-learn.org/stable/
    - *scipy 1.2.0* https://www.scipy.org/
    - *pandas 0.23.4* https://pandas.pydata.org/

2.3.Dekompozīcijas apraksts

2.3.1. Objektu dekompozīcija

PPS prasības tiek dalītas izmantojot objektorientētu dekompozīciju. Katra izstrādes procesā izveidotā klase tiek uzskatīta par objektu. Objekts var saturēt citus objektus, funkcijas, atribūtus, inicializāciju. Objekta inicializācija izveido pašu objektu. Funkcijas ir darbību instrukcijas, kuru rezultātā objekts viec kādu darbību, piemēram, tiek izmainīta kāda atribūta vērtība. Objekts, ko satur cits objekts, ir apakšobjekts. Atribūtam ir nosaukums un tas var saturēt apakšobjektu, skaistlisku vērtībī, simbolu virkni, sarakstu, vārdnīcu un citus datu tipus. Izveidot, atgriezt ogjekta instanci nozīmē izveidot objektu izsaucot objekta inicializācijas funkciju. Objekts un klase ir savstarpēji aizstājami termini.

Projekta un implementācijas pārskatāmības dēļ projektā sastopamās klases tiks iedalītas trijās kategorijās, pēc klašu nozīmes un darbības mērķa, piederības kādam *MMTA* procesa solim. Katra kategorija satur klases, kas atbilst šī procesa soļa darbībai. Klašu piederība kategorijām ir sekojoša:

- Palaišana
    - *TopicModeling*
    - *Trainer*
- Konfigurācijas ielase
    - *RunConfiguration*
    - *CorExConfiguration* -
- Teksta Analīze
    - *FileHelper* - Failu rakstīšana, lasīšana
    - *CorExExtractor* - Ekstraktors
    - *Factory* - Saskarne ar konfigurācijas ielasītājiem un ekstraktoriem

Klases aprakstošās tabulas satur funkciju nosaukumus un to aprakstus, klases atribūtu nosaukumus un atribūtu aprakstus.

Katra klase tiek izveidota atsevišķā failā.

Izstrādātas tika *CorExExtractor* un *CorExConfiguration* klases, kas seko *MMTA* koda struktūrai. Tika veikts darbs arī jau iepriekš izveidotajās klasēs *RunCofiguration, Factory, Trainer.*

2.3.1.1. Palaišana

Kategorija satur klases, kas atbild par projekta darbības uzsākšanu un darbības organizāciju, izmantojot citu klašu funkcijas.

- Palaišana
  - *TopicModeling* - *MMTA* sākuma punkts
  - *Trainer* - Ekstraktora vadīšana
  - *Factory* - Saskarne ar konfigurācijas ielasītājiem un ekstraktoriem
- Konfigurācijas ielase
  - *RunConfiguration* - Palaišanas konfigurācijas ielasītājs
  - *CorExConfiguration* - Ekstraktora konfigurācijas ielasītājs
- Teksta Analīze
  - *FileHelper* - Failu rakstīšana, lasīšana
  - *CorExExtractor* - Ekstraktors

2.3.1.1.1. TopicModeling

2.1 tabula

***MMTA* galvenais sākumpunkts. Pamatklase.**

Atribūti	**Apraksts**
trainer	Satur *Trainer* klases instanci.
Funkcijas	**Apraksts**
main()	Izveido *Trainer* klases instanci, izsauc *Trainer* klases funkciju *start_service()*

2.3.1.1.2. Trainer

2.2 tabula

Ekstraktora vadīšana

Atribūti	**Apraksts**
factory	Satur klases *Factory* instanci

extractor	Satur konfigurācijas failā definētā ekstraktora klases instanci (*CorExExtractor*)
extractorParameters	Satur sarakstu ar ekstraktora konfigurācijas faila parametriem
Funkcijas	**Apraksts**
init()	Klases inicializācija Izveido *Factory* klases instanci un saglabā to atribūtā *factory* Izsauc atribūta *factory* sekojošās funkcijas: • *create_run_configuration()* • *create_extractor_configuration()*
start_service()	Izsauc *Trainer* klases funkciju *run_training()*
run_training()	Izsauc atribūta *factory* funkciju *create_extractor()* Izsauc atribūta *extractor* sekojošās funkcijas: • *train()* • *extract()* • *save_models()*

2.3.1.1.3. Factory

2.3 tabula

Klase, kas atbild par saskarni ar konfigurācijas ielasītājiem, ekstraktoriem

Atribūti	**Apraksts**
modelTypes	Satur informāciju par to, kā saucas katra pieejamā ekstraktora klase un ekstraktora konfigurācijas ielsīšanas klase.
Funkcijas	**Apraksts**
init()	Klases inicializācija.
create_run_configuration()	Atgriež *RunConfiguration* klases instanci
create_extractor_configuration()	Atgriež konfigurācijas failā definētā ekstraktora konfigurācijas ielasītāju (*CorExConfiguration*)
create_extractor()	Atgriež konfigurācijas failā definētā ekstraktora klases instanci (*CorExExtractor*)

2.3.1.2. Konfigurācijas ielase

Klases, kas atbild par konfigurācijas faila sadaļu ielasīšanu

2.3.1.2.1. RunConfiguration

2.4 tabula

Klase palaišanas konfigurācijas ielasīšanai.

Atribūti	**Apraksts**
topicExtractory	Konfigurācijas failā definētais ekstraktora tips (simbolu virkne)
resultsFolder	Konfigurācijas failā definētā izvaddatu (rezultātu) mape (simbolu virkne)
dataFolder	Konfigurācijas failā definētā ievaddatu mape (simbolu virkne)
dataFileName	Konfigurācijas failā definētais ievaddatu (teksta dokumentu kopas) fails (simbolu virkne)
dataDelimiter	Konfigurācijas failā definētā ievaddatu (teksta dokumentu kopas) faila atdalošais simbols (simbols)
dataFileHasHeader	Galvenes esamība konfigurācijas failā definētajā ievaddatu (teksta dokumentu kopas) failam.
dataTextColumn	Tekstu saturošās kolonnas numurs (sākot ar 0) konfigurācijas failā definētajā ievaddatu (teksta dokumentu kopas) failam.
Funkcijas	**Apraksts**
init()	Klases inicializācija. Ielasa no konfigurācijas faila attiecīgās vērtības un saglabā tās kā klases atribūtus.

2.3.1.2.2. CorExConfiguration

2.5 tabula

Izstrādātā klase ekstraktora konfigurācijas ielasīšanai.

Atribūti	**Apraksts**
numOfTopics	Konfigurācijas failā definētais tēmu skaits (vesels skaitlis)

numOfIter	Konfigurācijas failā definētais iterāciju skaits (vesels skaitlis)
numOfKeywords	Konfigurācijas failā definētais atslēgvārdu skaits izejas failam *topicsByKeywords.csv* (vesels skaitlis)
probabilityThreshold	Konfigurācijas failā definētā pārliecības sliekšņvērtība izejas failam *multilabelData.csv* (skaitlis no 0-1)
maxFeatures	Konfigurācijas failā definētais maksimālais iezīmju skaits pielietotā vektorizētāja (*CountVectorizer*[]) izveidē (vesels skaitlis)
applyStemmingOrLemmatization	Konfigurācijas failā definētā izvēle starp vārdu apstrādi. Nekāda, vārdu celmošana vai vārdu lemmatizēšana. (*None, stemming* vai *lemmatization*)
removeStopwords	Konfigurācijas failā definētā izvēlne nebūtisko vārdu izņemšanas procesam. Neveikt vai veikt. (*True* vai *False*)
stopwordsFileName	Konfigurācijas failā definētā ievaddatu mapē atrodamā nebūtisko vārdu sarakstu saturošā faila nosaukums. (*None* vai simbolu virkne)
Funkcijas	**Apraksts**
init()	Klases inicializācija. Ielasa no konfigurācijas faila attiecīgās vērtības un saglabā tās kā klases atribūtus.

2.3.1.3. Teksta analīze

Klases, kas atbild par tiešu ievaddatu un izvaddatu apstrādi.

2.3.1.3.1. CorExExtractor

2.6 tabula

Ekstraktora klase

Atribūti	**Apraksts**
config	Satur vārdnīcu ar ekstraktora parametriem
docTopicDist	Satur *CorExModel.transform()* izvadīto matricu *p_y_given_x*[3] *(*saīsinājums no: *probability of Y for a given X*[7], nozīme:satur matricu, kur kolonnas atbilst tēmām, rindas

	atbilst teksta dokumentiem un vērtības ir decimāldaļskaitļi lielumā 0 – 1, kas apzīmē izveidotā modeļa pārliecību par katra dotā dokumenta piederību katrai dotai tēmai, kur 0 nozīmē pilnīgu nepiederību un 1 nozīmē pilnīgu piederību konkrētai tēmai*)* mainīgajā *docTopicDist;*
topics	Satur *CorExExtractor* klases funkcijas *get_topic_results()* atgriezto sarakstu
labeledDocuments	Satur *CorExExtractor* klases funkcijas *extract()* izveidoto divdimensionālu sarakstu ar trim kolonām: 1. teksta dokumenti, 2. tēmas numurs, ar lielāko pārliecību dotajam dokumentam, 3. konkrētās tēmas pārliecības lielums;
multilabelData	Satur *CorExExtracotr* klases funkcijas *extract()* izveidoto divdimensionālu sarakstu ar mainīgu kolonu skaitu katrā rindā: 1. teksta dokumenti, 2. tēmas numurs, ar lielāko pārliecību dotajam dokumentam, kas pārsniedz lietotāja uzstādīto *probabilityThreshold* 3. tēmas numurs, ar nākamo lielāko pārliecību dotajam dokumentam, kas pārsniedz lietotāja uzstādīto *probabilityThreshold.* Kolonnu skaits pēc pirmās kolonnas atkarīgs no tā, cik dotās rindas dokumentam katras tēmas pārliecības vērtība pārsniedz *probabilityThreshold.* Tik, cik pārsniedz, tik kolonnas pēc pirmās tiek pievienotas dotajai rindai Ja neviena tēmas pārliecības vērtība nepārsniedz *probabilityThreshold,* tad rinda satur tikai pirmo kolonnu.
stemmer	Var saturēt vārdu celmotāja SnowballStemmer instanci no bibliotēkas nltk
lemmatizer	Var saturēt vārdu lemmatizētāja WordNetLemmatizer instanci no bibliotēkas nltk
stopWordList	var saturēt nebūtisko vārdu sarakstu

termFrequency	Satur vektorizētus teksta dokumentus
countVectorizer	Satur vektorizētāja *CountVectorizer*[9] instanci
corExModel	Satur *corextopic*[3] instanci
featureNames	Satur teksta dokumentu datu kopas unikālo vārdu sarakstu alfabētiskā secība.
modelFileNames	Satur sarakstu ar failu nosaukumiem.
CONFIG_TYPES	Satur informāciju par parametru obligāto datu tipu un to noklusējuma vērtībām.
Funkcijas	**Apraksts**
init()	Klases inicializācija. Aizpilda sekojošos atribūtus: • ar tukšu sarakstu: *docTopicDist, topicWordDist, topics, labeledDocs, multilabelData,* documentsProcessed • *ar None: stemmer, lemmatizer, stopWordList, seedList, termFrequency, countVectorizer, corExModel, featureNames* • *config* atribūtu ar sarakstu, kas aizpildīts ar klases *Factory* funkcijas *create_extractor_configuration()* ielasītajām vērtībām no konfigurācijas faila. • *modelFileNames* satru sekojošu sarakstu: "*topicsByKeywords.csv, labeledDocuments.csv, topicScores.json, multilabelData.csv*" • *CONFIG_TYPES* satur šādu vārnīcu: o *numOfTopics*: ((int, type(None)), 10), o *numOfIter*: ((int, type(None)), 200), o *numOfKeywords*: ((int, type(None)), 10), o *probabilityThreshold*: ((float, type(None)), 0), o *maxFeatures*: ((int, type(None)), 5000), o *applyStemmingOrLemmatization*: ((str, type(None)), 'None'), o *removeStopwords*: ((bool, type(None)), 'True'), o *stopwordsFileName*: ((str, type(None)), 'None'),

	Kur, ieraksts srakstā sākas ar mainīgā nosaukumu, seko ar datu tipu, tad alternatīvo datu tipu un tad ar noklusējuma vērtību. Piemēram, "*numOfTopics: ((int, type(None)), 10),"* nozīmē, ka mainīgajam *numOfTopics* jābūt veselam skaitlim vai arī *None,* taču, ja ir *None*, tad vēlākā punktā *numOfTopics* atribūtam *CorExExtractor* klases funkcija *train(),* izmantojot klases *CorExExtractor* funkciju *check_config_valid(),* piešķirs noklusējuma vērtību 10.
train()	Secīgi izsauc klases *CorExExtrakctor* šādas funkcijas: *1. check_config_valid()* *2. load_stemmer_or_lemmatizer()* *3. load_stopwords()* *4. preprocess_text()* *5. calculate_term_frequency()* 6. *train_with_corex()*
check_config_valid()	Veic atribūta konfigurācijas parametru pārbaudi, salīdzinot *config* vārdnīcas vērtības ar *CONFIG_TYPES* atribūtā definētajām parametru raksturīpašībām
load_stemmer_or_lemmatizer()	Ja parametra *applyStemmingOrLemmatization* vērtība ir*:* 1. simbolu virkne "stemming", izveidot *snowballStemmer* instanci no *nltk* bibliotēkas. 2. simbolu virkne "lemmatization", izveidot *wordNetLemmatizer* instanci no *nltk* bibliotēkas. 3. *None,* tad turpināt darbu.
load_stopwords()	Ja *removeStopwords* vērtība ir: 1. *False,* tad turpināt darbu ar tukšu *stopWords* sarakstu 2. *True,* tad atribūtā *stopWordList* ielasīt sekojošās vērtības: ja nebūtisko vārdu saraksta faila nosaukums, kas definēts konfigurācijas failā ir: a) netukša simbolu virkne, tad izveidojot *FileHelper* klases instanci, un izmantojot tās funckiju *read_csv(),*

	ielasīt nebūtisko vārdu saraksta faila saturu, kas definēts konfigurācijas failā. b) vērtība *None* vai tukša simbolu virkne, tad ielasīt nebūtisko vārdu sarakstu no *nltk* bibliotēkas moduļa *stopwords.*
preprocess_text()	1. Izveidot klases *FileHelper* instanci, mainīgajā *documents* ielasīt teksta dokumentus, izmantojot *FileHelper* klases funkciju *read_csv()*; 2. Iterējot cauri mainīgā *documents* rindām, katrā iterācijā izmantojot funckiju *CorExExtractor.turn_text_into_tokens()* tokenizēt *documents* rindu un izveidotos tokenus(vārdus) apvienot starp tiem ievietojot atstarpi (izveidot "teikumu") un pievienot sarakstam *documentsProcessed.*
calculate_term_frequency()	1. Izmantojot bibliotēkas *CountVectorizer* funkciju *fit_transform* uz atribūtu *listDocuments*, aizpilda atribūtu *termFrequency;* 2. Izmantojot bibliotēkas *scipy* funkciju *csr_matrix* apstrādā atribūtu *termFrequency;* 3. Izmantojot *countVectorizer* funkciju *get_feature_names* aizpilda atribūtu *featureNames;*
train_with_corex()	1. Izveido *CorExModel* klases instanci (mašīnmācīšnās modeli) izmantojot *corextopic*[3] bibliotēkas iekšējo funkciju c*orex()*[3] *(*tālāk dokumentā funkcija *corextopic*[3] funkcijas tiks sauktas par klases *CorExModel* funkcijām*)* 2. Izmantojot klases *CorExModel* funkciju *fit()*[3] veic mašīnmācīšnās modeļa pielāgošanu (*fitting*) teksta dokumentu datu kopai
extract()	1. Izsauc klases *CorExModel* funkciju *transform()*

	2. Atribūtā *docTopicDist* aglabā klases *CorExModel* funkcijas *transform()*[3] izvadīto matricu *p_y_given_x*[3] (saīsinājums no: *probability of Y for a given X*[7], nozīme:satur matricu, kur kolonnas atbilst tēmām, rindas atbilst teksta dokumentiem un vērtības ir decimāldaļskaitļi lielumā 0 – 1, kas apzīmē izveidotā modeļa pārliecību par katra dotā dokumenta piederību katrai dotai tēmai, kur 0 nozīmē pilnīgu nepiederību (0%) un 1 nozīmē pilnīgu piederību (100%) konkrētai tēmai) 3. Iterējot cauri *docTopicDist* aizpilda atribūtus: a) *labeledDocs* – matrica ar trim kolonām. Pirmā kolonna satur dokumentu tekstus, otrā - tēmas numurs, ar lielāko pārliecību dotajam dokumentam, trešā – konkrētās tēmas pārliecības lielums; *b) multilabelData* – matrica ar neregulāru kolonnu skaitu katrā rindā. Katrai rindai ir vismaz viena kolonna. Pirmā kolonna – satur dokumentu tekstus. Otrā kolonna – tēmas numurs, ar lielāko pārliecību dotajam dokumentam, kas pārsniedz lietotāja uzstādīto *ProbabilityThreshold* (pārliecības sliekšņvērtība) konfigurācijas faila sadaļā COREX c) *docTopicScores* – vārdnīca (*dictionary*), kuras atslēga (*key*) ir katra dokumenta kārtas numurs, un katras atslēgas vērtība (*value*) ir saraksts (*array*) ar apakšsarakstiem (*nested array*), kuri satur divas vērtības: pirmā - tēmas numurs, otrā – modeļa pārliecība par doto tēmu dotajam dokumentam. Katra saraksta apakšsaraksti ir izkārtoti secībā pēc katra apakšaraksta otrās vērtības (pārliecības) dilstošā secībā.
get_topic_results()	Atgriež klases *CorExModel* funkcijas *get_topics()*[3] atgriezto sarakstu.

save_models()	1. Saglabā atribūtu *labeledDocs* failā ar nosaukumu *labeledDocuments.csv* izmantojot klases *FileHelper* funkciju *save_to_csv()* 2. Saglabā atribūtu *multilabelData* failā ar nosaukumu *multilabelData.csv* izmantojot klases *FileHelper* funkciju *save_to_csv()* 3. Saglabā mai atribūtu nīgo *docTopicScores* failā ar nosaukumu *topicScores.json* izmantojot klases *FileHelper* funkciju *save_json()* 4. Saglabā atribūtu *topics* failā ar nosaukumu *topicsByKeywords.csv* izmantojot klases *FileHelper* funkciju *save_to_csv()*
turn_text_into_tokens()	1. Ievadītajam tekstam noņem punktuāciju (pieturzīmes) 2. Katru dokumentu sadala vārdos (tokenizē) 3. Tokena lielos burtus nomaina uz mazajiem burtiem 4. Ja notiek nebūtisko vārdu izņemšana, tad tokenu izdzēš 5. Ja ir izvēlēta lematizēšana vai celmošana, tad tokenam veic attiecīgo apstrādi 6. Atgriež sarakstu ar tokeniem.

2.3.1.3.2. CorExModel

2.7 tabula

***Corextopic* bibliotēkas klase.**

Atribūti	**Apraksts**
n_hidden	Vērtība vienāda ar konfigurācijas failā norādīto *numOfTopics*
max_iter	Vērtība vienāda ar konfigurācijas failā norādīto *numOfIter*
words	Vērtība vienāda ar klases *CorExExtractor* atribūtu *featureName,* izmantots, lai varētu atgriezt dokumentus vektorizēšanas darbības rezultātu.
p_y_given_x	Atribūta nosaukums no: *probability of Y for a given X*[7], nozīme:satur matricu, kur kolonnas atbilst tēmām, rindas atbilst

	teksta dokumentiem un vērtības ir decimāldaļskaitļi lielumā 0 – 1, kas apzīmē izveidotā modeļa pārliecību par katra dotā dokumenta piederību katrai dotai tēmai, kur 0 nozīmē pilnīgu nepiederību un 1 nozīmē pilnīgu piederību konkrētai tēmai
Funkcijas	**Apraksts**
corex()	Skatīt c*orextopic*[3] bibliotēkas atsaucē atsauci
fit()	Skatīt c*orextopic*[3] bibliotēkas atsaucē atsauci
transform()	Skatīt c*orextopic*[3] bibliotēkas atsaucē atsauci

2.3.1.3.3. FileHelper

2.8 tabula

Klase, kas atbild par failu rakstīšanu, lasīšanu

Funkcijas	**Apraksts**
init()	Klases inicializācija
read_csv()	Ielasa *.csv* formāta failu mainīgajā *fileContents* un atgriež to.
save_to_csv()	Izmantojot *python* iebūvēto funkciju *csv* datu kopu ieraksta *.csv* formāta failā
save_pickle()	Izmantojot *python* iebūvēto funkciju *pickle* datu kopu ieraksta *.pkl* formāta failā
save_json()	Izmantojot *python* iebūvēto funkciju *json* datu kopu ieraksta *.json* formāta failā

2.3.1.3.4. BaseExtractor

2.9 tabula

Klase, kas atbild par ekstraktora klases sturktūras pareizību

Funkcijas	**Apraksts**
init()	Klases inicializācija
load_models()	Pārbauda tādas paša nosaukuma funckijas esamību *CorExExtracor* klasē
save_models()	Pārbauda tādas paša nosaukuma funckijas esamību *CorExExtracor* klasē

train()	Pārbauda tādas paša nosaukuma funckijas esamību *CorExExtracor* klasē
extract()	Pārbauda tādas paša nosaukuma funckijas esamību *CorExExtracor* klasē

3. TESTĒŠANAS DOKUMENTĀCIJA

3.1. Automātiskie vienībtesti un funkcionālie testi

Sistēmas *MMTA* izmaiņu pieteikuma gadījumā tiek veikti attālināti automātiski testi izmantotjot *Azure DevOps* izveidotos *pipelines* (konveijerapstrādes elements). Taču pirms izstrādātājs veic izmaiņu pieteikumu, tam jāveic ir lokālā vienībtestēšana uz ierīces, uz kuras ir veikta izstrāde, izmantojot *Python* 3.5.5 iekšējo vienībtestešanas satvaru (*framework*) ar nosaukumu *unittest*, komandrindā ierakstot *python -m unittest* komandu *(https://docs.python.org/3.5/library/unittest.html)*, ar komandrindu atvērtajā mapē tiek atrasti un palaisti visi faili, kā skripti (skat. vārdnīcu), kas satur *unittest* moduli.

```
(acn_malta_topics_build_7) C:\Users\mikus.kalnins\source\repos\Topic-Modeling>python -m unittest
----------------------------------------------------------------------
Ran 122 tests in 504.590s

OK

(acn_malta_topics_build_7) C:\Users\mikus.kalnins\source\repos\Topic-Modeling>
```

Att 1.6.1.1 ***MMTA*** **vienībtestestēšana izmantojot komandrindu**

3.2. Vienībtestēšanas rezultāti

Pēc *corextopic*[3] bibliotēkas implementācijas tika izveidoti lokālie vienībtesti, kas pārbaudītu *MMTA* sistēmai jaunpievienoto fukcionalitāti, pārbaudot katra faila darbību zemāk redzamajās tabulās. Pēc šo vienībtestu sekmīgas iziešanas, tika palaisti arī pārējie *MMTA* testi, lai pārbaudītu sistēmas integritāti ar *MMTA* sistēmu kopumā, kas sākotnēji nebija sekmīgi, taču pēc izmaiņu veikšanas gan implementācijā, gan testu papildināšana, rezultāti beidzot bija apmierinoši. Zemāk tiks aplūkoti implementācijas pārbaudīšanai izstrādātie testi, un veiktās izmaiņas, papildinājumi pārējos *MMTA* testos, jo pēc pasūtītāja lūguma, ļauts dokumentēt tikai *corextopic*[3] implementācijas ietvaros izstrādātos testus.

Tā kā katrs tests tiek palaists kā skripts un tiek veidots no atsevišķām funkcijām un atsevišķām klasēm ar klases funkcijām un klases atribūtiem, tad katra atsevišķa funkcija un klase tiks aprakstīta atsevišķi. Klases un to funkcijas un atribūti tiks aprakstīti līdzīgi kā PPA 2.3.1 nodaļā "Objektu dekopozīcija" jeb izmantojot objektus. Atsevišķās funkcijas savukārt tiks aprakstītas līdzīgi kā PPS 1.3.2 nodaļā "Funkcionālās prasības". Klases var izsaukt atsevišķās funkcijas, kas

atrodas dotajā testā, kā arī savrīgi ir pieminēt, ka atsevišķās funkcijas ir kodētas katra testa augšējā daļā jeb pirms klasēm. Tātad katra testa apraksts satur divas daļas:

- Funkcijas – atsevišķās funkcijas, apraksta veids balstīts uz PPS 1.3.2 nodaļu "Funkcionālās prasības";
- Klases – atsevišķās klases un to apraksta veids balstīts uz PPA 2.3.1 nodaļu "Objektu dekopozīcija".

Rezultāts: pēc izstrādes veikšanas un vienībtestu palaišanas, viss atbislt testu prasībām.

3.2.1. TestCorExExtractor

Tests pārbauda PPA sadaļā 2.3.1.3.1. "CorExExtractor" aprakstītās klases darbību

3.2.1.1. Funkcijas

3.1 tabula

Set_up_mock_corex_training_data()

Nosaukums: set_up_mock_corex_training_data()
Mērķis
Izveidot ievaddatu teksta dokumtu datu kopas *.csv* failu ievaddatos norādītajā mapē
Ievaddati
• *dataFolderPath* – mape, kurā saglabāt teksta dokumtu datu kopas *.csv* failu
Apstrāde
1. Izveidot un aizpildīt mainīgo *trainingData*, kas ir saraksts, kura apkšsaraksti ir teksta dokumenti; 2. Izmantojot *Python 3.5.5* bibliotēku *csv* izveidot mapē *dataFolderPath* failu ar nosaukumu "*data.csv",* kas saturēs *trainingData* mainīgā saturu.
Izvaddati
• nav
Kļūdu paziņojumi
• nav

3.2 tabula

Nosaukums: set_up_mock_corex_models()
Mērķis
Izveidot ievadītajā mapē *modelFolder* tos failus, kuri ir atzīmēti ar vērtību *True,* piemēram, ja *icludeCv* ir *True*, tad izveidot failu "*corExCv.pkl"*, kas aprakstīts 6. apstrādes punktā.
Ievaddat
• *modelFolder,* simbolu virkne; • *includeSettings,* Būla datu tips, noklusējuma *True;* • *icludeCv,* Būla datu tips, noklusējuma *True*; • *includeTf,* Būla datu tips, noklusējuma *True*; • *includeModel,* Būla datu tips, noklusējuma *True*; • *includeDocTopicDist,* Būla datu tips, noklusējuma *True*; • *includeTopicWordDist,* Būla datu tips, noklusējuma *True*; • *includeTopics,* Būla datu tips, noklusējuma *True*; • *includeLabeledDocs,* Būla datu tips, noklusējuma *True.*
Apstrāde
1. Ja *includeSettings,* tad izveido un aizpilda mainīgo *corExConfigValues* ar testa vārnīcu, kas satur cieti kodētas *CorEx* ekstraktora konfigurācijas testa vērtības. Saglabā mapē ar nosaukumu, kas vienāds ar *modelFolder,* failā ar nosaukumu "corExSetting.json". 2. Izveido un aizpilda mainīgo *mockData,* kas satur teksta dokumentu kopas sarakstu, 3. Izveido un aizpilda mainīgo *mockCvValue* (saīsinājums no *"mock count vecotrizer value"*) ar inicializētu *CountVecotrizer*[9] instanci, 4. Izveido un aizpilda mainīgo *mockTfValues* (saīsinājums no *"mock term frequency values"*) vektorizējot mainīgo *mockData* ar vektorizētāja instances *mockCvValue* funkciju *fit_transform()*[9], 5. ja *includeTf* (saīsinājums no *"include term frequency"*) vērtība ir *True,* tad *mockTfValues* saturu saglabā mapē ar nosaukumu, kas vienāds ar *modelFolder,* failā ar nosaukumu "*corExTf.pkl"*

6. ja *includeCv* (saīsinājums no *"include Count Vectorizer"*) vērtība ir *True*, tad *mockCvValues* saturu saglabā mapē ar nosaukumu, kas vienāds ar *modelFolder,* failā ar nosaukumu "*corExCv.pkl"*
7. ja *includeModel* vērtība ir *True,* tad izveido un aizpilda mainīgo *mockModelValues* izmantojot *corextopic*[3] bibliotēkas funckciju *corex()*[3] *corextopic*[3] klases instances izveidei un funkciju *fit*()[3] izveidotās instances (mašinmācīšanās modeļa pielāgošana) uz mainīgā *mockTfValues*. Beigās mainīgā *mockModelValues* saturu saglabā mapē ar nosaukumu, kas vienāds ar *modelFolder,* failā ar nosaukumu "*corExModel.pkl"*
8. ja *includeDocTopicDist* (saīsinājums no *"include document - topic distribution"*) vērtība ir *True,* tad izveido un aizpilda mainīgo *docTopicDistValues* ar testa vērtībām un mainīgā *docTopicDistValues* saturu saglabā mapē ar nosaukumu, kas vienāds ar *modelFolder,*" failā ar nosaukumu "*corExDocumentTopicMatrix.pkl"*
9. ja *includeTopicWordDist* (saīsinājums no *"include topic – word distribution"*) vērtība ir *True,* tad izveido un aizpilda mainīgo *topicWordDistValues* ar testa vērtībām un mainīgā *topicWordDistValues* saturu saglabā mapē ar nosaukumu, kas vienāds ar *modelFolder,*" failā ar nosaukumu "*corExTopicWordMatrix.pkl"*
10. ja *includeTopics* vērtība ir *True,* tad izveido un aizpilda mainīgo *topicsByKeywordsValues* ar testa vērtībām un mainīgā *topicsByKeywordsValues* saturu saglabā mapē ar nosaukumu, kas vienāds ar *modelFolder,* failā ar nosaukumu "*corExTopicsByKeywords.pkl"*
11. ja *includeLabeledDocs* (saīsinājums no *"labeled documents"*) vērtība ir *True,* tad izveido un aizpilda mainīgo *labeledDocsValues* ar testa vērtībām un mainīgā *labeledDocsValues* saturu saglabā mapē ar nosaukumu, kas vienāds ar *modelFolder,* failā ar nosaukumu "*corExLabeledDocuments.pkl"*

3.2.1.2. Klases

3.3 tabula

TestCorExExtractorSetUp

Funkcijas	**Apraksts**
setUp()	neko

test_initialize_corex_extractor()	izveido *CorExExtractor* instanci ar tukšiem parametriem un pārbauda vai: 1. izveidotā ektraktora tips ir ar nosaukumu “CorEx” 2. pārbauda vai ekstraktorā esoša mainīgā *docTopicDist* saturs ir tukšs 3. pārbauda vai ekstraktorā esoša mainīgā *modelFileNames* saturā ir 12 elementi 4. pārbauda vai ekstraktorā esoša mainīgā *CONFIG_TYPES* saturā ir 11 elementi
tearDown()	neko

3.4 tabula

TestCorExExtractorWithoutExistingModel

Funkcijas	**Apraksts**
setUp()	1. Inicializē klasi 2. Izveido pagaidu mapi un saglabā tās atršanās vietu mainīgajā *tempDir*. 3. Izveido un aizpilda mainīgo *trainingData*, ko saglabā failā ar nosaukumu “data.csv” mapē *tempDir*. 4. Izveido un aizpilda mainīgo *mockStopwordsData*, ko saglabā failā ar nosaukumu “mock_stopwords.csv” mapē *tempDir*. 5. Izveido mainīgo *extractorParams* un aizpilda to ar testa vērtībām. 6. Mainīgajā *extractor* izveido *CorExExtractor* klases instanci izmantojot *extractorParams* vērtības.
test_training_without_stemming	Uzstāda *extractor.config.applyStemmingOrLemmatization* uz *None*, izmanto *extractor.train(),* pārbauda vai sakrīt *self.extractor.countVectorizer.vocabulary_* sakrīt ar *expectedVocabulary,* kas ir cieti iekodēta šajā funkcijā.

test_training_with_stemming	Līdzīgi kā *test_training_without_stemming,* bet ar *extractor.config.applyStemmingOrLemmatization* vērtību uzstādītu uz "*stemming*"
test_turn_text_into_tokens	Pārbauda vai *extractor.turn_text_into_tokens* funkcija izmet kļūdas paziņojumu uz skaitliskas vērtības, bet atgriež gaidīto rezultātu uz funkcijai padotā dotā teksta.
test_remove_punctuation	Pārbauda vai *extractor.test_remove_punctuation* funkcija izmet kļūdas paziņojumu uz skaitliskas vērtības, bet atgriež gaidīto rezultātu par funkcijai padoto tekstu.
test_save_models	Pārbauda vai *extractor.test_remove_punctuation* funkcija izmet kļūdas paziņojumu uz skaitliskas vērtības, bet izveido gaidītos failus, ja fukcijai padota *tempDir* apvienojums ar *"results".*
test_load_stemmer	Pārbauda vai ar ar konkrētiem uzstādījumiem tiek izmestas kļudas paziņojumi, bet ar pareizajiem vai izsaucot extractor.load_stemmer_or_lemmatizer extractor.stemmer satur *SnowballStemmer* instanci un vai extractor.lemmatizer satur *None*
test_load_lemmatizer	Pārbauda vai ar ar konkrētiem uzstādījumiem tiek izmestas kļudas paziņojumi, bet ar pareizajiem vai izsaucot extractor.load_stemmer_or_lemmatizer extractor.lemmatizer satur *WordNetLemmatizer* instanci un vai extractor.stemmer satur *None*
test_load_stopwords	• Pārbauda vai ar uzstādījumu.*removeStopwords = False* atribūts *extractor.stopWordList* vienāds ar *None* • Pārbauda vai ar uzstādījumu.*removeStopwords = False* un *stopwordsFileName = None* atribūts *extractor.stopWordList* ir *set* instance • Pārbauda vai ar uzstādījumu.*removeStopwords = False* un *stopwordsFileName = mock_stopwords.csv* atribūts *extractor.stopWordList* ir *set* datu tipa instance

test_train_with_corex	• Pārbauda vai izmet kļūdu paziņojumu, ja trūkst kāds no ievaddatiem izsaucot funkciju *extractor.train_with_corex()*
test_check_config_valid	• Pārbauda vai ar izveidotu nepareiz parametru tiek izmests kļūdu paziņojums • Un vai ar nepareiza tipa parametru vērtībām arī tiek izmests kļūdu paziņojums
tearDown()	Izdzēš *tempDir* mapi un visus failus un datus, kas ir bijuši saglabāti *tempDir* mapē

3.5 tabula

TestCorExExtractorWithExistingModel

Funkcijas	**Apraksts**
setUp()	1. Inicializē klasi 2. Izveido pagaidu mapi un saglabā tās atršanās vietu mainīgajā *tempDir.* 3. Izveido mainīgo *extractorParams* un aizpilda to ar testa vērtībām. 4. Mainīgajā *extractor* izveido *CorExExtractor* klases instanci izmantojot *extractorParams* vērtības. 5. Pagaidu mapē izveido mapi “results” un saglabā mainīgajā *resultsFolder;* 6. Pagaidu mapē izveido mapi “data” un saglabā mainīgajā *dataFolder;* 7. Pagaidu mapē izveido mapi “models” un saglabā mainīgajā *modelFolder* 8. Izsauc testa funkciju *set_up_mock_corex_training_data()* padododod *dataFolder* 9. Izsauc testa funkciju *set_up_mock_corex_models()* padododod *modelFolder*

	10. Izsauc funkciju *extractor.load_models()* padodod *modelFolder* un saglabājot instanci mainīgajā *corExExtractor*. 11. Izveido un aizpilda *extractor* instances sekojošus mainīgos: o *labeledDocs* ar cieti kodētām vērtībām o *docTopicScores* ar cieti kodētām vērtībām o *documents* ar cieti kodētām vērtībām o *docTopicDist* ar cieti kodētām vērtībām o *topicWordDist* ar cieti kodētām vērtībām o *termFrequency* ar mainīgā *corExExtractor* atribūtu *termFrequency* o *corExModel* ar mainīgā *corExExtractor* atribūtu *corExModel*
test_load_models	• Pārbauda vai *load_models()* uz neeksistējošas mapes izmet kļūdu ziņojumu • Pārbauda, vai netiek izmesti kļūdu ziņojumi uz *modelFolder*
test_extract	• Pārbauda vai *extractor* instance ir *CorExExtracotr* instance • Pārbauda vai *extractor.extract()* neizmet kļūdu paziņojumus
test_train_with_corex	• Pārbauda vai tiek imzesta kļūda ar pārlieku liela parametra *maxFeatures* lielumu izsaucot funkciju *train_with_corex()* • Pārbauda vai netiek izmesti kļūdu paziņojumi, ja ar parametra *maxFeatures* lielums ir atiblstošs izsaucot funkciju *train_with_corex()*
test_save_models	• Izmanto *CountVectorizer* funkciju uz *extractor* atribūtu *documents*, izmanto *save_models* uz *resultsFolder* un pārbauda vai ir izveidoti vajadzīgie faili.
test_multilabelData_is_list	• Pārbauda vai *multilabel* atribūts ir *list* tipa

test_multilabelData_list_content	• Pārbauda vai *multilabel* atribūta saturs tiek veidots pareizi izsaucot *extract()* pārbauda un salīdzinot rezultātu ar funkcijā ģenerēto testa sarakstu.
test_multilabelData_csv_content	• Pārbauda vai *multilabel* tiek pareizi saglabāts *.csv* formātā, veicot *extract(), save_models(),* ielasot izveidoto *.csv* failu un salīdzinot tā saturu ar funkcijā ģenerētu testa sarakstu.
tearDown()	Izdzēš *tempDir* mapi un visus failus un datus, kas ir bijuši saglabāti *tempDir* mapē

3.6 tabula

TestCorExExtractorWithExistingModelWithoutSettings

Funkcijas	**Apraksts**
setUp()	1. Inicializē klasi 2. Izveido pagaidu mapi un saglabā tās atršanās vietu mainīgajā *tempDir.* 3. Izveido mainīgo *extractorParams* un aizpilda to ar testa vērtībām. 4. Mainīgajā *extractor* izveido *CorExExtractor* klases instanci izmantojot *extractorParams* vērtības. 5. Pagaidu mapē izveido mapi "results" un saglabā mainīgajā *resultsFolder;* 6. Pagaidu mapē izveido mapi "data" un saglabā mainīgajā *dataFolder;* 7. Pagaidu mapē izveido mapi "models" un saglabā mainīgajā *modelFolder* 8. Izsauc testa funkciju *set_up_mock_corex_training_data()* padodod *dataFolder* 9. Izsauc testa funkciju *set_up_mock_corex_models()* padodod *modelFolder* un *includeSettings* ar vērtību *False*
test_load_models()	• Pārbauda, vai izsaucot *load_models()* tiek izmest kļūdas paziņojums, ka kāds fails nav atrasts

tearDown()	Izdzēš *tempDir* mapi un visus failus un datus, kas ir bijuši saglabāti *tempDir* mapē

3.7 tabula

TestCorExExtractorWithExistingModelWithoutTF

Funkcijas	**Apraksts**
setUp()	1. Inicializē klasi 2. Izveido pagaidu mapi un saglabā tās atršanās vietu mainīgajā *tempDir*. 3. Izveido mainīgo *extractorParams* un aizpilda to ar testa vērtībām. 4. Mainīgajā *extractor* izveido *CorExExtractor* klases instanci izmantojot *extractorParams* vērtības. 5. Pagaidu mapē izveido mapi "results" un saglabā mainīgajā *resultsFolder;* 6. Pagaidu mapē izveido mapi "data" un saglabā mainīgajā *dataFolder;* 7. Pagaidu mapē izveido mapi "models" un saglabā mainīgajā *modelFolder* 8. Izsauc testa funkciju *set_up_mock_corex_training_data()* padodod *dataFolder* Izsauc testa funkciju *set_up_mock_corex_models()* padodod *modelFolder* un *includeTf* ar vērtību *False*
test_load_models()	Pārbauda, vai izsaucot *load_models()* tiek izmest kļūdas paziņojums, ka kāds fails nav atrasts
tearDown()	Izdzēš *tempDir* mapi un visus failus un datus, kas ir bijuši saglabāti *tempDir* mapē

3.8 tabula

TestCorExExtractorWithExistingModelWithoutCorExModel

Funkcijas	**Apraksts**
setUp()	1. Inicializē klasi

	2. Izveido pagaidu mapi un saglabā tās atršanās vietu mainīgajā *tempDir*. 3. Izveido mainīgo *extractorParams* un aizpilda to ar testa vērtībām. 4. Mainīgajā *extractor* izveido *CorExExtractor* klases instanci izmantojot *extractorParams* vērtības. 5. Pagaidu mapē izveido mapi "results" un saglabā mainīgajā *resultsFolder;* 6. Pagaidu mapē izveido mapi "data" un saglabā mainīgajā *dataFolder;* 7. Pagaidu mapē izveido mapi "models" un saglabā mainīgajā *modelFolder* 8. Izsauc testa funkciju *set_up_mock_corex_training_data()* padodod *dataFolder* Izsauc testa funkciju *set_up_mock_corex_models()* padodod *modelFolder* un *includeModel* ar vērtību *False*
test_load_models()	Pārbauda, vai izsaucot *load_models()* tiek izmest kļūdas paziņojums, ka kāds fails nav atrasts
tearDown()	Izdzēš *tempDir* mapi un visus failus un datus, kas ir bijuši saglabāti *tempDir* mapē

3.9 tabula

TestCorExExtractorWithExistingModelWithoutCV

Funkcijas	**Apraksts**
setUp()	1. Inicializē klasi 2. Izveido pagaidu mapi un saglabā tās atršanās vietu mainīgajā *tempDir*. 3. Izveido mainīgo *extractorParams* un aizpilda to ar testa vērtībām. 4. Mainīgajā *extractor* izveido *CorExExtractor* klases instanci izmantojot *extractorParams* vērtības.

	5. Pagaidu mapē izveido mapi “results” un saglabā mainīgajā *resultsFolder;* 6. Pagaidu mapē izveido mapi “data” un saglabā mainīgajā *dataFolder;* 7. Pagaidu mapē izveido mapi “models” un saglabā mainīgajā *modelFolder* 8. Izsauc testa funkciju *set_up_mock_corex_training_data()* padodod *dataFolder* Izsauc testa funkciju *set_up_mock_corex_models()* padodod *modelFolder* un *includeCv* ar vērtību *False*
test_load_models()	Pārbauda, vai izsaucot *load_models()* tiek izmest kļūdas paziņojums, ka kāds fails nav atrasts
tearDown()	Izdzēš *tempDir* mapi un visus failus un datus, kas ir bijuši saglabāti *tempDir* mapē

3.10 tabula

TestCorExExtractorWithExistingModelWithoutDocTopicDist

setUp()	1. Inicializē klasi 2. Izveido pagaidu mapi un saglabā tās atršanās vietu mainīgajā *tempDir.* 3. Izveido mainīgo *extractorParams* un aizpilda to ar testa vērtībām. 4. Mainīgajā *extractor* izveido *CorExExtractor* klases instanci izmantojot *extractorParams* vērtības. 5. Pagaidu mapē izveido mapi “results” un saglabā mainīgajā *resultsFolder;* 6. Pagaidu mapē izveido mapi “data” un saglabā mainīgajā *dataFolder;* 7. Pagaidu mapē izveido mapi “models” un saglabā mainīgajā *modelFolder* 8. Izsauc testa funkciju *set_up_mock_corex_training_data()* padodod *dataFolder*

	Izsauc testa funkciju *set_up_mock_corex_models()* padodod *modelFolder* un *includeDocTopicDist* ar vērtību *False*
test_load_models()	Pārbauda, vai izsaucot *load_models()* tiek izmest kļūdas paziņojums, ka kāds fails nav atrasts
tearDown()	Izdzēš *tempDir* mapi un visus failus un datus, kas ir bijuši saglabāti *tempDir* mapē

3.11 tabula

TestCorExExtractorWithExistingModelWithoutTopicWordcDist

Funkcijas	**Apraksts**
setUp()	1. Inicializē klasi 2. Izveido pagaidu mapi un saglabā tās atršanās vietu mainīgajā *tempDir*. 3. Izveido mainīgo *extractorParams* un aizpilda to ar testa vērtībām. 4. Mainīgajā *extractor* izveido *CorExExtractor* klases instanci izmantojot *extractorParams* vērtības. 5. Pagaidu mapē izveido mapi "results" un saglabā mainīgajā *resultsFolder;* 6. Pagaidu mapē izveido mapi "data" un saglabā mainīgajā *dataFolder;* 7. Pagaidu mapē izveido mapi "models" un saglabā mainīgajā *modelFolder* 8. Izsauc testa funkciju *set_up_mock_corex_training_data()* padodod *dataFolder* Izsauc testa funkciju *set_up_mock_corex_models()* padodod *modelFolder* un *includeTopicWordDist* ar vērtību *False*
test_load_models()	Pārbauda, vai izsaucot *load_models()* tiek izmest kļūdas paziņojums, ka kāds fails nav atrasts
tearDown()	Izdzēš *tempDir* mapi un visus failus un datus, kas ir bijuši saglabāti *tempDir* mapē

3.12 tabula

TestCorExExtractorWithExistingModelWithoutTopics

Funkcijas	**Apraksts**
setUp()	1. Inicializē klasi 2. Izveido pagaidu mapi un saglabā tās atršanās vietu mainīgajā *tempDir*. 3. Izveido mainīgo *extractorParams* un aizpilda to ar testa vērtībām. 4. Mainīgajā *extractor* izveido *CorExExtractor* klases instanci izmantojot *extractorParams* vērtības. 5. Pagaidu mapē izveido mapi "results" un saglabā mainīgajā *resultsFolder;* 6. Pagaidu mapē izveido mapi "data" un saglabā mainīgajā *dataFolder;* 7. Pagaidu mapē izveido mapi "models" un saglabā mainīgajā *modelFolder* 8. Izsauc testa funkciju *set_up_mock_corex_training_data()* padodod *dataFolder* Izsauc testa funkciju *set_up_mock_corex_models()* padodod *modelFolder* un *includeTopics* ar vērtību *False*
test_load_models()	Pārbauda, vai izsaucot *load_models()* tiek izmest kļūdas paziņojums, ka kāds fails nav atrasts
tearDown()	Izdzēš *tempDir* mapi un visus failus un datus, kas ir bijuši saglabāti *tempDir* mapē

3.13 tabula

TestCorExExtractorWithExistingModelWithoutLabeledDocs

Funkcijas	**Apraksts**
setUp()	1. Inicializē klasi 2. Izveido pagaidu mapi un saglabā tās atršanās vietu mainīgajā *tempDir*.

	3. Izveido mainīgo *extractorParams* un aizpilda to ar testa vērtībām. 4. Mainīgajā *extractor* izveido *CorExExtractor* klases instanci izmantojot *extractorParams* vērtības. 5. Pagaidu mapē izveido mapi "results" un saglabā mainīgajā *resultsFolder;* 6. Pagaidu mapē izveido mapi "data" un saglabā mainīgajā *dataFolder;* 7. Pagaidu mapē izveido mapi "models" un saglabā mainīgajā *modelFolder* 8. Izsauc testa funkciju *set_up_mock_corex_training_data()* padododot *dataFolder* Izsauc testa funkciju *set_up_mock_corex_models()* padododot *modelFolder* un *includeLabeledDocs* ar vērtību *False*
test_load_models()	Pārbauda, vai izsaucot *load_models()* tiek izmest kļūdas paziņojums, ka kāds fails nav atrasts
tearDown()	Izdzēš *tempDir* mapi un visus failus un datus, kas ir bijuši saglabāti *tempDir* mapē

3.2.2. TestCorExConfiguration

Tests pārbauda PPA sadaļā 2.3.1.2.2. "CorExConfiguration" aprakstītās klases darbību

3.2.2.1. Funkcijas

Atsevišķu funkciju testam nav.

3.2.2.2. Klases

3.14 tabula

TestCorExParameters

Atribūti	**Apraksts**

Funkcijas	Apraksts
setUp()	1. Inicializē klasi 2. Izveido pagaidu mapi un saglabā tās atršanās vietu mainīgajā *tempDir*.
test_number_of_corex_variables()	• Izveido un aizpilda mainīgo *expectedCorExVariables*, kas satur cieti kodētu sarakstu ar parametru nosaukumiem, kurus sagaidām, ka tik ielasīti izmantojot *CorExConfiguration* klasi. Veic pārbaud vai ielasītais saturs sakrīt ar mainīgā *expectedCorExVariables* saturu
test_create_corex_configuration _with_incorrect_values()	• Izveido konfigurācijas failu, kas satur tikai COREX sadaļu ar nepareiziem datu tipiem un pārbauda, vai ielasot izmantojot *CorExConfiguration* klasi, netiek paturētas nepareizās vērtības
test_create_corex_configuration _with_correct_values()	Izveido konfigurācijas failu, kas satur tikai COREX sadaļu ar pareiziem datu tipiem un pārbauda, vai ielasot izmantojot *CorExConfiguration* klasi, tiek paturētas pareizās vērtības
tearDown()	Izdzēš *tempDir* mapi un visus failus un datus, kas ir bijuši saglabāti *tempDir* mapē

3.2.3. TestFactory

Tests pārbauda PPA sadaļā 2.3.1.1.3. “Factory” aprakstītās klases darbību

3.2.3.1. Funkcijas

Atsevišķu funkciju testam nav.

3.2.3.2. Klases

3.15 tabula

SetUpFactoryTest

Atribūti	Apraksts

Funkcijas	**Apraksts**
setUp()	nekas nenotiek
test_initialize_factory()	• Inicializē klasi *Factory* un pārbauda, vai tās atribūts *MODEL_TYPES* ir ar pareizu saturu salīdzinot pret cieti kodētu vārdnīcu
tearDown()	nekas nenotiek

3.16 tabula

CorExFunctionalFactoryTest

Atribūti	**Apraksts**
Funkcijas	**Apraksts**
setUp()	1. Inicializē klasi 2. Izveido pagaidu mapi un saglabā tās atršanās vietu mainīgajā *tempDir* 3. Izveido un aizpilda mainīgo *CONFIG_TYPES* 4. Izveido un aizpilda mainīgo *corExConfigValues* 5. Saglabā *tempDir* mapē *corExConfigValues* mainīgā vērtību failā "corExSettings.json" 6. Izveido un aizpilda mainīgo *mockTfValues* 7. Saglabā *tempDir* mapē *mockTfValues* mainīgā vērtību failā *"corExTf.pkl"* 8. Izveido un aizpilda mainīgo *mockCvValue* 9. Saglabā *tempDir* mapē *mockCvValue* mainīgā vērtību failā *"corExCv.pkl"* 10. Izveido un aizpilda mainīgo *mockModelValues* 11. Saglabā *tempDir* mapē *mockModelValues* mainīgā vērtību failā "*corExModel.pkl"* 12. Izveido un aizpilda mainīgo *docTopicDistValues* 13. Saglabā *tempDir* mapē *docTopicDistValues* mainīgā vērtību failā "corExDocumentTopicMatrix.pkl"

	14. Izveido un aizpilda mainīgo *topicWordDistValues* 15. Saglabā *tempDir* mapē *topicWordDistValues* mainīgā vērtību failā "corExTopicWordMatrix.pkl" 16. Izveido un aizpilda mainīgo *labeledDocsValues* 17. Saglabā *tempDir* mapē *labeledDocsValues* mainīgā vērtību failā "corExLabeledDocuments.pkl" 18. Izveido un aizpilda mainīgo *topicsByKeywordsValues* 19. Saglabā *tempDir* mapē *topicsByKeywordsValues* mainīgā vērtību failā ar nosaukumu "corExTopicsByKeywords.pkl" 20. Izveido *Factory* klases instanci un saglabā to mainīgajā *factory* 21. Izveido un aizpilda mainīgo *configFile* ar *"./TopicModeling.cfg"*
test_create_run_configuration()	• Izveido un aizpilda mainīgo *runParams* izmantojot *factory* instances funkciju *create_run_configuration* uz faila *configFile* un pārbauda vai *runParams* ir *RunConfiguration* instance
test_create_extractor_configuration()	• Izmantojot *factory* instances funkciju *create_extractor_configuration()* ar nepareizām vērtībām, pārbauda, vai tiks izsaukts kļūdu paziņojums • Izveido un aizpilda mainīgo *corExParams* izmantojot *factory* instances funkciju *create_extractor_configuration()* padodod failu *configFile* un ar simbolu virkni "CorEx" pārbauda vai *corExParams* ir *CorExConfiguration* instance
test_create_extractor()	• Izmantojot *factory* instances funkciju *create_extractor()* ar nepareizām vērtībām, pārbauda, vai tiks izsaukts kļūdu paziņojums

	• Izveido un aizpilda mainīgo *corExParams* izmantojot *factory* instances funkciju *create_extractor()* padodod failu *configFile* un ar simbolu virkni "CorEx" un izveido un aizpilda mainīgo *corExExtractor* izmantojot *factory* instances funkciju *create_extractor()* padodod tai simbolu virkni "CorEx" un corExParams. Beigās pārbauda vai *corExExtractor* ir *CorExExtractor* instance
tearDown()	Izdzēš *tempDir* mapi un visus failus un datus, kas ir bijuši saglabāti *tempDir* mapē

4. PROJEKTA ORGANIZĀCIJA

Papildu funkciju izstrāde mašīnmācīšanās teksta analizatoram (turpmāk *MMTA*) tika organizēta pēc spējās izstrādes apakškategorijas *Scrum* metodoloģijas. Uzsākot projektu tika veiktas sarunas ar pasūtītāju, lai noskaidrotu kādai papildus funkcionalitātei jātiek izstrādātai. Kad tika noskaidrots, ka jāveic ir *corextopic*[3] bibliotēkas implementācija, tad tika veikta iepazīšanās ar *MMTA* sistēmu kopumā un apskatīta un izprasta iepriekšējo *MMTA* izstrādātāju pieeja citu bibliotēku implementācijai. *MMTA* izprašanas laikā tika veikti mazi uzdevumi, un, ar katru izpildīto uzdevu, tika izpildītas pasūtīja prasības, kā arī uzstādītas jaunas prasības un jauni uzdevumi. Pēc katra uzdevuma izpildes tuvojoties pilnīgākai projekta izpildei.

Izstrādes process tika sadalīts sākumā divu, pēc tam triju nedēļu garos intervālos, ko sauc par sprintiem, katra sprinta sākuma tika veikta sprinta plānošana un katra sprinta beigās tika veikts sprinta atskats (*retrospective*) kopā ar iesaistītajām pusēm un komandas biedriem. Katras dienas sākumā tika veiktas *Scrum* tikšanās ar komandas biedriem, kuras laikā katrs komandas biedrs izstāstīja padarīto, sastaptās grūtības vai kādus citus jaunumus, kas saistīti ar *MMTA*. Visas tikšanās notika ar video zvana palīdzību izmantojot *Microsfot Teams* lietotni.

Izstrādes pārvaldība notika izmantojot *Azure DevOps* lietotni, kas uzlaboja darba organizāciju un pārskati. Uzsākot izstrādi, tika noteikts, ka programmprodukta koda izstrādes termiņš ir 2019. gada 17. maijs, pēc kā jāseko dokumentēšanas procesam līdz 2019.gada 25. maijam.

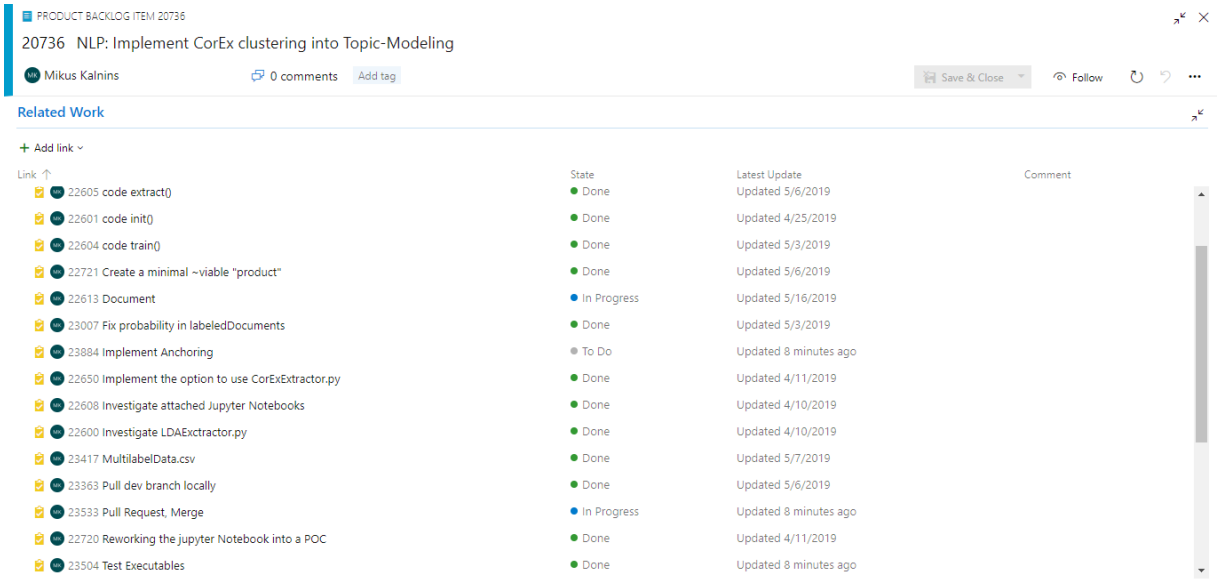
Patērētā laika un līdzekļu mazināšanai tika izmantoti atvērtā pirmkoda rīki, par kuriem lasāms PPA nodaļā "*Tehniskie risinājumi*".

Izstrāde veikta izstrādes vidēs *Microsoft Visual Studio Professional 2017 15.9.12, Jupyter Notebook 4.4.0* un *Sublime Text Version 3.2, Build 3200*. Versiju kontrole un izstraādāta projekta augšuppielāde notika izmantojot *Azure DevOps* un *git-gui version 0.21.GITGUI* ar *git version 2.20.1.windows.1*.

Projekta dokumentēšana tika veikta izmantojot *Microsoft Office 365* lietotni *Microsoft Word* un interneta lietotni *Drawio*[8] .

Projekta dokumentēšanas, plānošanas un organizēšanas laikā ņemts vērā rakstītais LU Datorikas fakultātes e-studiju kursa materiālos no kursiem "*Kvalifikācijas darbs*" un

"*Programminženierija*". Darba noformēšanā ievērots rīkojums Nr. 1/69 "*Prasības noslēguma darbu (bakalaura, maģistra darbu, diplomdarbu un kvalifikācijas darbu) izstrādāšanai un aizstāvēšanai Latvijas Universitātē*".



Att 4.1 **Uzdevumu pārvalde izmantojot *Azure DevOps* platformu.**

5. KONFIGURĀCIJU PĀRVALDĪBA

CorEx[7] implementācijas konfigurāciju pārvaldība tika īstenota ar versiju vadību. Tādā veidā dodot iespēju sekot līdzi izmaiņām programmprodukta kodā un dokumentācijā: Šāda veida konfigurāciju pārvaldība ļāva brīvi mainīt sistēmas versijas un tās zarus, ja tika atklātas novirzes kādā no implementācijas daļām.

Versiju pārvaldība īstenota ar *Git* repozitoriju, ko pārvalda izmantojot *git-gui* lietotni vai arī izmantojot izstrādes vidē *Microsoft Visual Studio Professional 2017* pieejamo *git* paplašinājumu. Repozitorijjs tika saglabāts *Azure DevOps*[2] platformā. Repozitoriju glabāšanas platformas izvēli veicis ir pasūtītājs

Jebkuras izstrādātāja veiktās dokumentācijas vai *MMTA* repozitorija izmaiņas tiek saglabātas pasūtītāja nodrošinātajā izstrādes ierīcē, izmaiņas tiek aprakstītas un kad izmaiņu veikšana ir beigta, tad tās tiek iesūtītas *Git* repozitorijā, un saglabāts *Azure DevOps*[2] platformā, lai izmaiņas varētu tikt notestētas, pārskatītas, pievienotas izstrādes zaram un pieejams citiem izstrādātājiem.

Aprakstītā konfigurāciju pārvaldība ir sevi attaisnojusi un ir visnotaļ efektīva gan viena, gan vairāku izstrādāju gadījumā. Lai gan protams, ja projekts būtu viena cilvēka izstrādāts, tad varētu iztikt bez *Azure DevOps*[2], bet gan izmantot, piemēram, *DropBox*.

6. KVALITĀTES NODROŠINĀŠANA

Kvalitātes nodrošināšana *CorEx*[7] implementācijas laikā tiek nodrošināta izpildot pasūtītāja izvirzītās prasībās un ieviešot prasībās minētās funkcionālitātes. Darba kvalitātes novērtēšana notiek *CorEx*[7] implementācijas gala produkta atbilstība pasūtītāja prasībām.

Zemāk redzams saraksts ar būtiskākajām kvalitātes nodrošināšanas aktivitātēms, kuras tika izpildītas darba veikšanas laikā. Daļa no aktivitātēm ir aprakstītas sīkāk šajā dokumentā. Saraksts nav uzskatāms par galīgu.

- Pasūtītāja *MMTA* sistēmas apguve
- Prasību definēšana (skatīt "*Programmatūras prasību specifikācija*")
- Plānošana (skatīt "*Projekta organizācija*")
- Dokumentēšanas izstrāde
- Tehniskā realizācija (skatīt "T*ehniskie risinājumi*")
- Vienībtestēšana (skatīt "*Testēšanas dokumentācija*")
- Konfigurāciju pārvaldība (sktīt "*Konfigurāciju pārvaldību*")
- Konsultācijas ar *MMTA* projekta kolēģiem
- Konsultācijas ar pasūtītāju
- Pasūtītāja nodrošinātas papildus apmācības
- Konsultācijas ar kvalifikācijas darba vadītāju
- Implementācijas rīka[3] studēšana
- Implementācijas rīkā izmantotās literatūras[7] studēšana

7. DARBIETILPĪBAS NOVĒRTĒJUMS

Darbietilpības rezultāts

Darba ietilpības aprēķins veikts izmantojot *Azure DevOps* un *Microsoft Excel*. Zemāk redzama tabula, kas satur datus par paveiktā darba izpildei patērēto darba dienu skaitu, pieņemot, ka viena darba diena ir astoņas stundas.

7.1 tabula

Patērēto darba dienu skaits izstrādes laikā

Paveiktais darbs	**Dienu skaits**
Dokumentācijas izstrāde	
Konsultācija ar pasūtītāju	10
Prasību specificēšana	10
Prasību specifikācijas saskaņošana	3
Projektējuma izveide	10
Konsultācijas ar kolēģiem	3
Projekta izstrādes plānošana	1
Corextopic izpēte	5
Testu dokumentēšana	3
Funkcionālo klašu programmēšana, vienībtestēšana	
Trainer	0.5
Factory	0.5
CorExConfiguration	5
CorExExtractor	20
Automātisko vienībtestu izstrāde, testēšana	
TestCorExExtractor	3
TestCorExConfiguration	2.5
TestFactory	0.5
KOPĀ	**77**

- Kopā patērētās dienas: 77
- Kopā patērētās darba stundas: 616
- Stundu skaits vienā personmēnesī: 152
- *CorEx*[7] implementācija veikta aptuveni: 4 personmēnešos

COCOMO darbietilpības mērījums

Izmantojot COCOMO[4][5] darbietilpības mērīšanas metodi tika aprēķināts, ka EAF[4] vienāds ar 1.15, un tā kā ir trīs vienkārši ievadi un trīs vienkārši izvadi, tad funkcijpunktu (FP) skaits ir vienāds ar 24[5]. Tā kā projekts tiek veidots izmantojot Python valodu, tad LOC/FP (LOC - *lines of code* jeb koda rindiņas) koeficents ir 67. Lai aprēķinātu rindiņu skaitu, sareizina FP ar LOC/FP, kas ir 24x67 = 1,608. Tātad KLOC (kiloLOC jeb tūkstots koda rindiņu) ir vienāds ar 1,6. Beigu beigās aprēķinot personmēnešu skaitu izmanto formulu a*(KLOC ^b)*EAF [4], kur a vienāds ar 3.0 un b ir 1.12, iegūst, ka pērsonmēnešu skaits ir **5.84** personmēneši.

NOBEIGUMS

CorEx[7] implementācija ir sekmīgi sekojot prasību specifikācijai un ievērjot uzstādīto terminu. Ir gan palicis izstrādātā projekta lietošana un lietotāju testēšana, kā arī ir iespējams veikt dažus uzlabojumus un papildinājums. Šādi papioldinājumi varētu būt *Corextopic*[3] bibliotēkā piedāvātās anchored trenēšanas implementācija, kas nodrošinātu rezultātu precīzāku iegūšanu sekojot lietotāja uzstādījumiem un papildus ievaddatiem.

Izstrādes laikā tika apgūts kā projekta pārvaldības un projekta dokumentācijas pamati. Grūtākais bija izprast, kā pasūtījam jau eksistējošā sistēma (*MMTA)* darbojās un kā izpildīt pasūtītāja prasības.

Tika iegūta nozīmīgi svarīga pieredze programmēšanas valodas *Python 3.5.5* pielietojumā, kā arī mašīnamācīšanās izstrādes pamatos, konkrētāk teksta analīzes sfērā. Kā arī tika iegūta pieredze veidojot vienībtestus izmantojot *Python 3.5.5* satvaru *unittest*.

Šobrīd turpinās *CorEx*[7] implementācijas uzlabojumi un funkcionāli papildinājumi, kā arī tiks veikti papildinājumi un citu funkcionalitāšu implementēšana *MMTA* sistēmā.

IZMANTOTĀ LITERATŪRA

Grāmatas

1. Pressman R. *Software engineering: a praticioner's approach.* New York: McGraw-Hill Higher Education, 2010.

Tīmekļa resursi

2. Azure DevOps [tiešsaiste] [atsauce 23.05.2019] Pieejams internetā: https://dev.azure.com
3. Corextopic [tiešsaiste] [atsauce 23.05.2019] Pieejams internetā: https://github.com/gregversteeg/corex_topic
4. Cocomo [tiešsaiste] [atsauce 23.05.2019] Pieejams internetā: https://en.wikipedia.org/wiki/COCOMO
5. Cocomo lietošanas piemērs [tiešsaiste] [atsauce 23.05.2019] Pieejams internetā: https://estudijas.lu.lv/mod/resource/view.php?id=128556
6. What is Scrum? [tiešsaiste] [atsauce 23.05.2019] Pieejams internetā: https://www.scrum.org/resources/what-is-scrum
7. Gallagher, Ryan J., Kyle Reing, David Kale, and Greg Ver Steeg. *"Anchored Correlation Explanation: Topic Modeling with Minimal Domain Knowledge."* Transactions of the Association for Computational Linguistics (TACL), 2017. pp.529-531 [tiešsaiste] [atsauce 23.05.2019] Pieejams internetā:https://www.transacl.org/ojs/index.php/tacl/article/view/1244,
8. Draw.io [tiešsaiste] [atsauce 23.05.2019] Pieejams internetā: https://www.draw.io/
9. CountVectorizer dokumentācija [tiešsaiste] [atsauce 23.05.2019] Internetā: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
10. Letonika [tiešsaiste] [atsauce 23.05.2019] Pieejams internetā: https://www.letonika.lv/groups/default.aspx?r=1107&q=v%C4%81rdn%C4%ABca&id=2071241&g=1

Latvijas Valsts standarti

11. Latvijas Valsts Standarts LVS 68:1996 "*Programmatūras prasību specifikācijas ceļvedis*",
12. Latvijas Valsts Standarts LVS 72:1996 *"Ieteicamā prakse programmatūras projektējuma aprakstīšanai"*

Kvalifikācijas darbs „***Papildu funkciju izstrāde mašīnmācīšanās teksta analizatoram***” izstrādāts Latvijas Universitātes Datorikas fakultātē.

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

Autors: ***Mikus Kalniņš*** **_____ ___.05.2019**.

Rekomendēju darbu aizstāvēšanai
Darba vadītājs/a: ***prof. Jānis Zuters*** **_____ ___.05.2019.**

Recenzents: ***Jānis Vempers, ZZ Dats SIA, projektu pārvaldnieks***

Darbs iesniegts 28.05.2018.
Kvalifikācijas darbu pārbaudījumu komisijas sekretārs: *Darja Solodovņikova*_____

Darbs aizstāvēts kvalifikācijas darbu pārbaudījuma komisijas sēdē
___.06.2019. prot. Nr. _____

Komisijas sekretārs(-e): _____