Mercari – price suggestion challenge

Bojana Đerić Ivan Emanuel Pavlov Tatjana Ramljak Lara Rajković

***Sažetak*—U ovom radu rješavali smo zadatak japanske tvrtke *Mercari* postavljen na *Kaggleu* u svrhu poboljšanja njihove aplikacije. Nama je glavni cilj bio isprobati kakve će rezultate dati različite metode naučene na kolegiju Strojno učenje i pri tome dati što točnije rješenje zadatka.**

I. Uvod

Mercari je japanska aplikacija za prodaju i kupnju stvari. Korisnici aplikacije mogu prodati bilo što, bilo korišteno ili ne: odjeću, igračke, elektronsku i sportsku opremu... Njihova misija je da prodaja stvari postane lakša od kupovine. Kako bi to postigli htjeli su da aplikacija predloži cijenu korisnicima kada stave novi predmet na prodaju.

U tu svrhu otvoreno je natjecanje na *Kaggleu*. Zadatak je predvidjeti cijenu predmeta s obzirom na informacije koje je korisnik unio o tom predmetu. Cilj ovog projekta je isprobati različite metode i modele za rješavanje danog problema te zatim usporediti dobivene rezultate te napraviti anasmbl modela.

II. Opis problema

Podaci su preuzeti s *Kagglea* i skup podataka za učenje sadrži 1 482 535 redova. Stupci koji sadrže podatke o svakom *listingu* su:

- `train_id` – *ID listinga*
- `name` – naziv *listinga*
- `item_condition_id` – u kakvom je stanju predmet (1 do 5)
- `category_name` – naziv kategorije
- `brand_name` – marka predmeta
- `shipping` – 1 ako je naknada plaćena od prodavača, 0 inače
- `item_description` – opis predmeta
- `price` – cijena, ciljna varijabla

Mjera pogreške modela je *RMSLE (Root Mean Squared Logarithmic Error)*. Ona se dobiva po formuli

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(log(p_i+1)-log(a_i+1))^2}$$

pri čemu su
n - broj podataka u skupu,
p_i - predviđene cijene,
a_i - stvarne cijene.

Iz skupa podataka za testiranje izdvojili smo 5% podataka za validaciju.

III. Metode i pristupi rješavanja problema

A. Predprocesuiranje teksta i LDA model

Za svaki model potrebno je `name` i `item_description` koji su obliku teksta procesuirati kako bismo izvukli bitne informacije. U tu svrhu tekstove smo tokenizirali, izbacili *stop words*, stvorili bigrame i trigrame te lematizirali pomoću paketa *spacy* i *gensim*.

Ako gledamo na opise predmeta kao dokumente koji su generirani različitim temama i pretpostavimo da je svaka tema generirana različitim riječima, ima smisla istrenirati *Latent Dirichlet Allocation (LDA)* topic model na tim dokumentima (nakon predprocesuiranja). *LDA* model pretpostavlja da znamo broj tema u dokumentima unaprijed. Pošto je teško unaprijed zaključiti koji je broj tema u opisima predmeta, prvo je istreniran model s 20 tema. Vizualnim prikazom dobivenih tema uvidjeli smo da se većina tih tema preklapa pa je sljedeći izabran broj tema 5. Na slikama 1 i 2 vidimo da se teme većinom ne preklapaju. Zbog predugog treniranja modela (više od 5 sati), nismo više mijenjali brojeve tema i uspoređivali *topic coherence score*, što je mjera za evaluaciju tog modela.

Za svaki dokument dobili smo distribuciju tema u dokumentu i tako dobivamo 5 novih značajki, vjerojatnosti da pojedini dokument pripada pojedinoj temi. Te značajke koristili smo u modelima navedenim u sljedećem odjeljku.

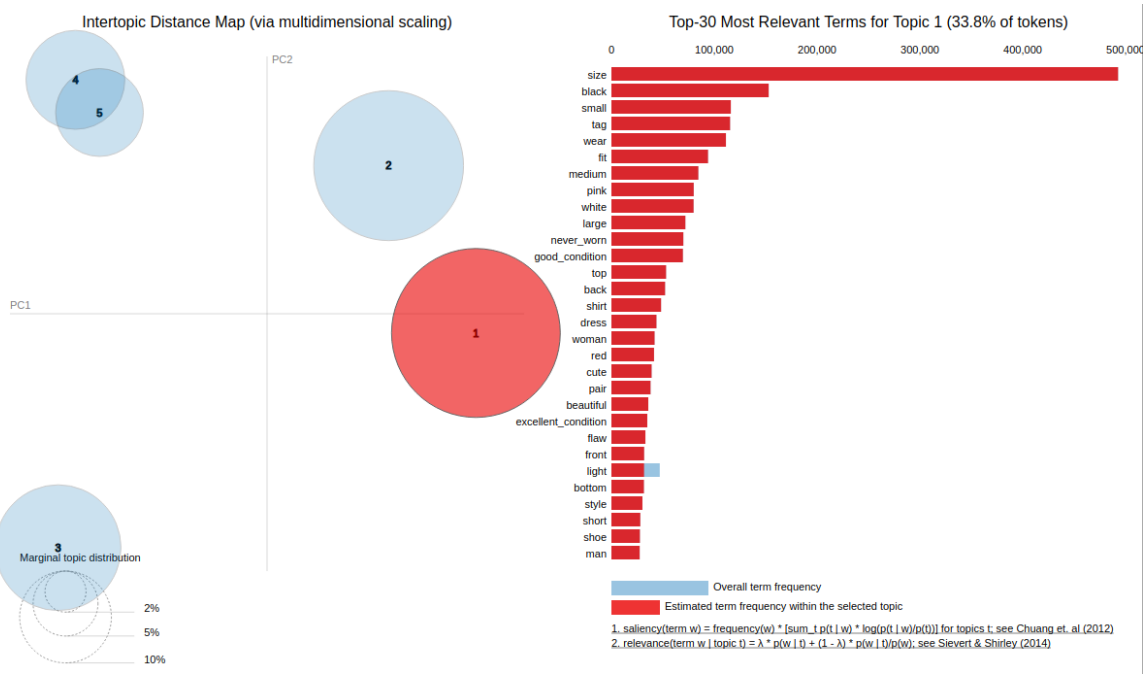
B. Ridge regresija, LightGBM, CatBoost

Budući da je ciljna varijabla neprekidna, prvi model kojeg smo odlučili isprobati bio je linearna regresija. Brzim pregledom rješenja na *Kaggleu* vidjeli smo da *Ridge* regresija daje najbolji rezultat pa smo se odlučili za nju. *Ridge* regresija rješava problem koreliranosti nezavisnih varijabli te ne radi selekciju varijabli za razliku od *Lasso* regresije.

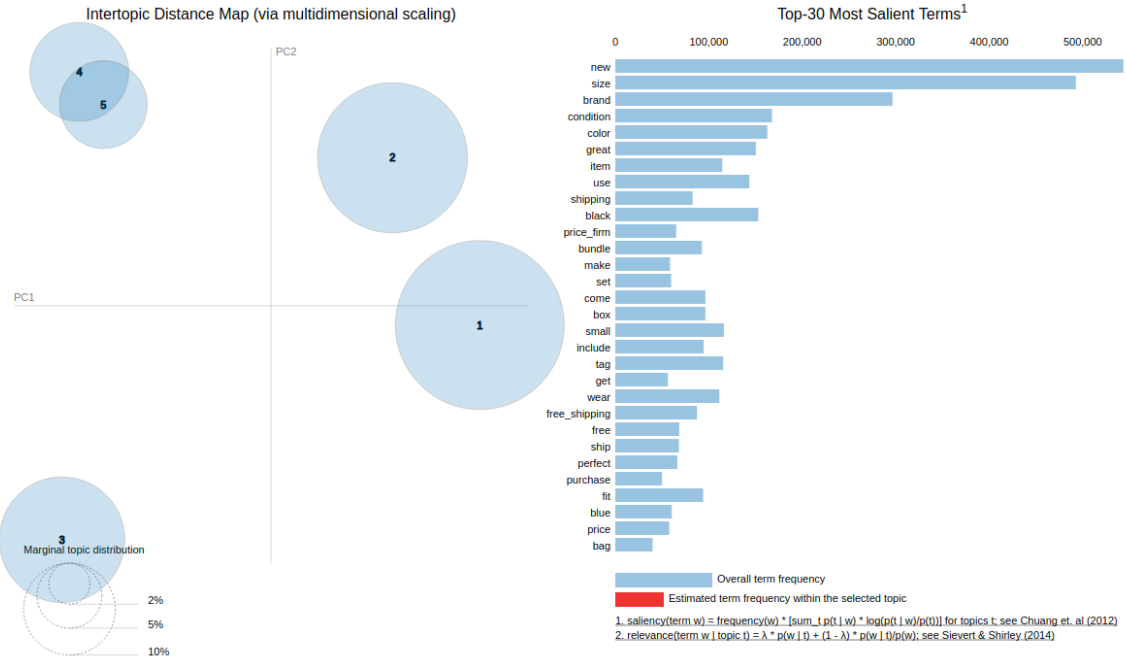
Nakon popunjavanja nedostajućih vrijednosti, rastavili smo `category_name` na tri kategorije. Za provedbu Ridge regresije `item_condition_id` i `shipping` smo pretvorili u *dummy* varijable (kodiranje pomoću nula i jedinica). Kategorije smo transformirali pomoću *CountVectorizera* te `brand_name` pomoću *LabelBinarizera* iz *sklearna*. Za `item_description` korišten je *TfidfVectorizer*, a za `name` isprobali smo i *TfidfVectorizer* i *CountVectorizer* te uspoređivali razliku. Sve dobivene matrice i vektore smo spojili u jednu rijetku matricu te na njoj proveli treniranje modela.

Istu matricu smo koristili za treniranje *LightGBMa*.

Zatim smo istrenirali *CatBoost* i *LightGBM* s drugim pristupom. Sve kategorijske varijable smo tretirali kao takve bez obrade, dodali smo vjerojatnosti tema koje smo dobili iz *LDA*



Slika 1. Prikaz tema iz `item_description` u prostoru i frekvencija riječi iz teme 1

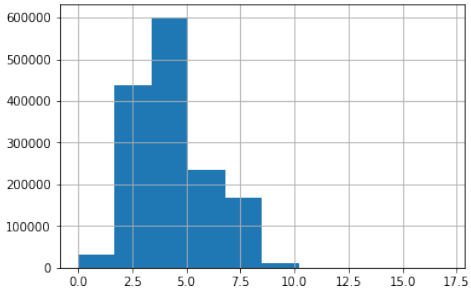


Slika 2. Prikaz tema iz `item_description` u prostoru i frekvencija svih riječi

modla te numeričke varijable koje smo dobili za `name` i `item_description` kao srednje vrijednosti *tf-idf* težina. U *CatBoost* možemo unijeti stringove kao kategorijske značajke dok smo za *LightGBM* pomoću *LabelEncodera* pretvorili stringove u brojeve. Oba modela izabrali smo zbog automatskog baratanja modela kategorijskim značajkama.

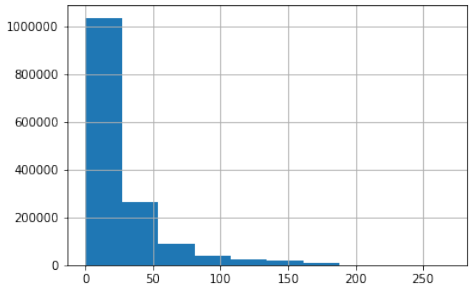
C. Neuronske mreže: predprocesuiranje za mrežu, treniranje mreže

Budući da se radi o problemu s relativno velikim skupom za treniranje s preko milijun primjera, neuronske mreže se čine kao izgledan kandidat za rješavanje problema. U skupu podataka na nekim mjestima nema unesenih vrijednosti što je logično jer se radi o realnom skupu podataka. Ljudi sami upisuju svoje podatke za proizvode koje prodaju pa neka mjesta ponekad nisu popunjena. Taj problem možemo riješiti na dva načina: jedna od njih je ne koristiti sve *listinge* u kojima se pojavljuju nedostajuće vrijednosti, a drugi način je upisati neku vrijednost kao npr. "missing" na sva mjesta gdje vrijednost nedostaje. Mi smo se odlučili za drugi način. Također, isprobali smo je li vrijedno rastavljati `category_name` na tri različite kategorije. Usporedili smo na jednostavnijim modelima, razlika *RMSLE* je bila u korist modela koji koristi rastavljanje na tri kategorije, iako ponešto usporava rad modela. S obzirom da raspolažemo kategorijskim i tekstualnim značajkama, potrebno ih je obraditi. Za obradu `category_name` i `brand_name` korišten je *LabelEncoder* iz *sklearn*a. Za obradu tekstualnih značajki koristili smo najjednostavniju metodu: tokenizacija *Kerasovim tokenizerom*. *Tokenizer* će naučiti tokene, tj. pridijeliti neku vrijednost svakoj riječi koju "vidi" u korpusu teksta na kojemu ga *fit*amo. Izlaz je niz brojeva u kojem svaka riječ koja se nalazi u rečenici dobiva svoj token, tj. neku vrijednost. Te sekvence nisu istih duljina pa je to potrebno nakadno obraditi prije korištenja u neuronskoj mreži. Potrebno je bilo obraditi `name` i `item_description`.



Slika 3. Distribucija broja riječi u `name`

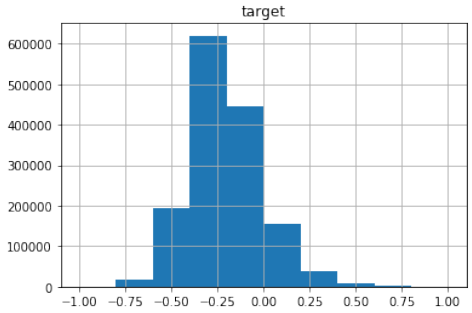
Iz distribucije vidimo da se broj riječi u imenu predmeta uglavnom kreće između 2-8 riječi. Za opis vidimo da je preko milijun opisa predmeta sadržano u 25 riječi. To znači da duljinu gornje opisanih nizova možemo ograničiti na duljinu od oko 8 riječi za ime, te 25 riječi za opis (malo više jer ipak je jedna trećina duljeg opisa). Odlučili smo ipak koristiti nešto veće brojeve kako bismo ograničili nešto veći skup, konkretno, za `name` smatrali smo da je optimalno uzeti 10,



Slika 4. Distribucija broja riječi u `item_description`

a za `item_description` 75. Koristeći *pad_sequences* iz *keras.preprocessing.sequence* namješteno je da svi nizovi budu jednakih duljina (dakle, svi nizovi za `name` su duljine 10).

U svrhu normaliziranja cijena, one su logaritmirane i skalirane da budu između -1 i 1.



Slika 5. `Log cijene`

Konačno, prije nego je započeto treniranje, podijelili smo *training set* na novi set za treniranje i set za validaciju korištenjem *train_test_split* iz *sklearn.model_selection* u omjeru 95:5.

Korišten je funkcionalni *API* model iz *Kerasa*. U modelu su prvo zadane dimenzije *inputa*. Potom su korišteni *embedding layeri* za `name`, `item_description`, `brand_name`, `category_name` i `item_condition`. Mijenjanjem dimenzije embeddinga za `brand_name`, `category_name` i `item_condition` layer, dobivaju se lošiji rezultati. To se ne čini sumnjivo jer se ionako radi o par riječi kao vrijednostima ovih varijabli. Za varijable `name`,`item_description` bi očekivali da ćemo povećavanjem dimenzije do neke granice imati bolje rezultate uz dulji trening, jer se povećava broj parametara, no rezultati nisu puno bolji.

Kod manje dimenzionalnosti *embeddinga*, mreža se istrenira nakon otprilike 5 epoha. U konačnici, zadnji model se trenira oko 15ak epoha, no uz još nekoliko promjena koje će sada biti navedene.-

S obzirom da su `name`,`item_description` tekstualne varijable *RNN layeri* su dobar kandidat za odabir aritekture. Počinjemo s 2 *GRU layera*, s brojem ćelija 8 i 16. Potom smo zamijenli *GRU layere* s *LSTM layerima* i dobili dosta lošije rezultate. Zaustavili smo se na 2 *GRU layera* s 10 i 20 ćelija.

Također, isprobali smo jednostavnu implementaciju *CNN 1d layera*, no nismo dobili zadovoljavajuće rezultate, ali pretpostavljamo da se daju dosta poboljšati daljnjom optimizacijom.

Koristili smo *Dropout* i većinom smo se držali omjera 0.1(udio koji se gasi). Povećavanjem *dropout* udjela smo pogoršali rezultate. Ekstreman slučaj gdje smo stavili omjer na 0.8 je odmah saturirao mrežu, tj. činilo se kao da se mreža ne trenira. Moguće jer je previše mrtvih neurona, pa se popravci u težinama ne propagiraju dobro.

Također, dodali smo 3 *dense layera* s aktivacijskom funkcijom *RELU*. Nismo previše isprobavali druge aktivacijske funkcije, no to se čini kao očiti korak u daljnjoj optimizaciji. Također, probali smo *Leaky RELU layere* te se čini da su oni inkrementalno utjecali.

Dense layeri su definitivno popravili rezultat. Počeli smo s brojem neurona u svakom *layeru* 100, te postepeno povećavali. Svaki inkrement nije naročito popravljao model, no kroz nekoliko inkremenata broja neurona i broj *layera*, dobili smo zadovoljavajući popravak rezultata. To nije začuđujuće jer smo očito time mreži dali dodatnu fleksibilnost kod učenja.

Izlazni *layer* je jedan *dense layer* s linearnom aktivacijskom funkcijom jer predviđamo neku kontinuiranu vrijednost. *Loss* funkcija koju smo koristili je *MSE*, dok je *optimizer* bio prvo *adam* pa *nadam*. *Nadam* daje nešto bolje rezultate. Također kroz trening su se računale dvije pogreške *Mean Absolute Error* i *Root Mean Squared Logarithmic Error*.

U konačnoj mreži smo imali 62 423 967 parametara i svi su se mogli trenirati.

D. Kombiniranje modela

Za dobivanje ansambla modela optimizirali smo sljedeću funkciju:

$$y_p = \sum_{i=1}^{5} w_i y_i$$

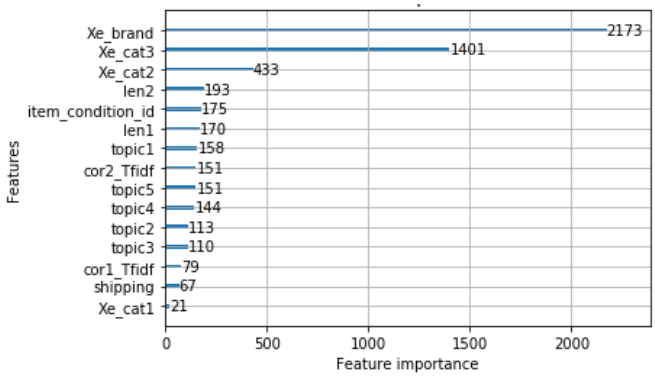
pri čemu su w_i težine takve da je $\sum_{i=1}^{5} w_i = 1$, y_i su predikcije pojedinih modela i y_p je konačna predikcija.

IV. Rezultati

Rezultati koje navodimo su izračunate vrijednosti *RMSLE* na validacijskom skupu. Mnoge od metoda koje smo htjeli isprobati i primijeniti za izvođenje su zahtijevale mnogo više od sat vremena što je bilo vremensko ograničenje na izvođenje koda na *Kaggleu*.

A. Ridge regresija, LightGBM, CatBoost

Najbolji rezultat od ovih modela daje *LightGBM* treniran na matrici u kojem je za procesuiranje `name` korišten *CountVectorizer* i on iznosi 0.46105. Slijedi *Ridge* regresija s rezultatom 0.47196 također trenirana na matrici koja je dala bolji rezultat kada je za `name` bio korišten *TfidfVectorizer*. Zatim slijedi *CatBoost* s rezultatom 0.52900 pa *LightGBM* s rezultatom 0.53438, koji su trenirani s kategorijskim varijablama. Dakle, rezultati su bolji kada smo ručno pretvarali kategorijske značajke u numeričke. Razlog tome je vjerojatno velik broj različitih kategorija [1]. Također u matrici smo zadržali sve *tf-idf* vrijednosti dok u posljednja dva modela imamo samo njihove prosječne vrijednosti, što također čini razliku. Na slici 6 vidimo važnosti značajki posljednjeg modela.



Slika 6. *Feature importance značajki LightGBMa*

B. Neuronske mreže

Početni *template* [2] daje na validacijskom skupu rezultat 0.48948. Manjim promjenama, kao što su smanjivanje *batch size* na 10.000 ili manje, povećanje broja epoha (nije uvijek rezultiralo smanjenjem greške, na nekim modelima baš suprotno), dodavanjem *dense layer*a s aktivacijom `relu` (iako su ispitane i opcije s aktivacijskim funcijama `sigmoid` i `tanh`) smanjujemo grešku. Dodavanje *dense layer*a se čini kao najbolji modifikator. Daljnje promjene su davale inkrementalne promjene uz cijenu duljeg treninga makar je i vrijeme treninga bilo relativno stabilno (nakon što smo dodali ekstra *embedding* dimenzije i *dense layer*e). Najbolji rezultati koji smo postigli bili su 0.44817 i 0.44212. Oba ta rezultata postignuta su na rekurentnim mrežama. Najbolji rezultat na konvolucijskim mrežama bio je 0.45740, dobiven dodavanjem četiri *dense layera*, što je ujedno bio i prvi značajniji pomak naprijed što se rezultata na konvolucijskim mrežama tiče.

C. Kombiniranje modela

Dobivene težine su: $w_1 = 0.011, w_2 = 0.199, w_3 = 0.036, w_4 = 0.008, w_5 = 0.516$ pri čemu su modeli redom:

- *LightGBM* s kategorijama
- *LightGBM* s matricom
- *Ridge* regresija
- *CatBoost*
- Neuronska mreža

Greška *RMSLE* iznosi 0.43254 što je najbolji rezultat koji smo dobili.

V. Osvrt na druge pristupe

Kako je problem preuzet s Kagglea, na stranici su ponuđena razna rješenja, među kojima su neka slična našima. Najbolja rješenja su ujedno i najkompleksnija, i samim tim su nama bila vrlo teška za pratiti i razumjeti. Navedimo neke metode i ideje koji su koristili drugi sudionici *challenge*a:

- pretrenirani *Word2vec* [3] (koji je dosta lošiji od gotovo svih naših rezultata)
- različite vrste ansambala (npr. RNN i Ridge regresija)
- kombinacija *Follow the Regularized Leader* algoritma, *Factorization Machines* i konvolucijskih mreža
- višedretvenost, gdje je primjenjena, dala je vrlo dobre rezultate, između 0.405 i 0.415.
- *CatBoost*, *LightGBM*, *XGBoost*
- zanimljiva R biblioteka *h2o*, s naglaskom na *AutoML: Automatic Machine Learning*

Primjetili smo da se u dosta *kernela* nije pridavalo toliko pažnje predprocesiranju koliko je bilo moguće, pretpostavljamo da je to slučaj zbog nužne uštede vremena. Jako se rijetko radilo popunjavanje `brand`ova, što smo mi, recimo, koristili.

VI. Moguć nastavak istraživanja

Značajke koje smo dobili iz *LDA* modela koristili smo u ostalim modelima, no treniranje *LDA* modela je bilo jako sporo te zbog toga nismo našli optimalan broj tema. Trebalo bi istražiti postoji li implementacija modela koji se trenira u kraćem roku ili način da se treniranje ubrza.

Također, trebalo bi optimizirati parametre za *Ridge* regresiju, *LightGBM* i *CatBoost* čime se nismo puno bavili jer smo htjeli isprobati više modela.

Nadalje, mogli bismo se baviti redukcijom dimenzije dobivene matrice korištene u regresiji te ispitivanjem interakcija između značajki.

U slučaju neuronskih mreža također postoji puno mjesta za napredak. Vidjevši da se stvari jako malo mijenjaju dodavanjem ekstra *layera*, ili povećavanjem dimenzionalnosti *embeddinga*, čini se da smo postigli aproksimativni limit s ovakvom mrežom. Dalji popravci bi bili: daljnje istraživanje rekurentnih mreža, razvijanje intuicije kada su one u pitanju, te konstruiranje kompleksnije arhitekture koja neće biti redundantna. Također pristup preko konvolucijskih mreža se čini kao izgledan kandidat, no mi nismo stigli probati i taj pristup, s obzirom da modeli nisu pokazivali toliko boje rezultate s obzirom na promjene kao modeli s rekurentnim mrežama. Naša obrada podataka je bila relativno rudimentarna. Dodatnim sređivanjem podataka bi vjerojatno otvorili mogućnosti iscrpiti puno više informacija iz danih podataka koristeći NN, a pogotovo druge metode. Koristili smo vlastite *embeddinge*, no zanimljiv eksperiment bi bio korištenje vanjskih *embeddinga* kao *GLOVE*, *Word2Vec*, no imali smo ograničeno vrijeme i samo smo načeli te pristupe (te smo imali problem s *hardwareom* jer su npr. *Glove embeddingsi* bili veliki). Također kako se u *NLP*-u razvija *transfer learning*, bilo bi zanimljivo probati neke pretrenirane modele kao *BERT*. Korištenjem uvodnih *layera* u mreži iz nekog jakog modela bi mogli istrenirati nekoliko finalnih *layera* da bolje predviđa cijene. Također, nismo koristili biblioteke kao *Spacy* (osim za *LDA* model) ili *NLTK* koje već imaju puno za ponuditi kad je obrada teksta u pitanju. No treba imati u vidu da je ovo natjecanje bilo ograničeno na 1 sat izvršavanja u *kernelu* pa bi svi ti kompliciraniji pristupi vjerojatno premašili limit.

Literatura

[1] LightGBM. (2019). Categorical Feature Support, adresa: https://lightgbm.readthedocs.io/en/latest/Advanced-Topics.html.

[2] 'noobhound'. (). A simple nn solution with Keras, adresa: https://www.kaggle.com/knowledgegrappler/a-simple-nn-solution-with-keras-0-48611-pl.

[3] (), adresa: https://www.kaggle.com/girlduck/item-description-cnn-word2vec.

[4] Mercari. (18. veljače 2018). Mercari Price Suggestion Challenge, adresa: https://www.kaggle.com/c/mercari-price-suggestion-challenge/.

[5] A. Géron. (2017). Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.

[6] F. Chollet. (2017). Deep Learning with Python.

[7] A. Papiu. (). Ridge script, adresa: https://www.kaggle.com/apapiu/ridge-script.