# 2. Seminarska naloga: poročilo

# Inteligentni sistemi

## 14. 1. 2024

## Uvod

To poročilo 2. seminarske naloge je delo Arjana Skoka in Vite Naglič. Cilj naloge je klasifikacija novic v njihove zvrsti z uporabo metod procesiranja naravnega jezika (NLP). Naloga je rešena v pythonu z uporabo več knjižnic, pretežno scikit-learn.

## Podatki in predpriprava besedil

Podatki so podani v json obliki. Vsak vnos predstavlja podatke o eni novici. Vsega skupaj imamo 148122 novic. Za vsako novico podanih več atributov, a nas najbolj zanimajo njen naslov, kratek opis in zvrst novice. Podan je tudi link do celotne novice, od koder bi lahko pobrali njeno besedilo, a je proces zelo počasen in se zaradi tega za uporabo tega nisva odločila. Manjkajočih je bilo zelo malo vnosov naslovov ali opisov. V teh primerih sva se odločila, da za manjkajoči del vstaviva prazen niz.

Naravni jezik sam po sebi ni primeren za obdelavo z scikitom. Zato ga moramo na to posebej pripraviti. Proces predpriprave je sestavljen iz naslednjih korakov:

1. Delitev stavkov na posamezne besede in transformacija vseh črk v male
2. Odstranitev ločil
3. Odstranitev pogostih besed, ki ne prispevajo ničesar k analizi besedila
4. Lematizacija ali korenenje besed, ki besede vrne v svojo osnovno obliko glede na pomen, da jih lahko obravnavamo kot eno samo (npr. running 🡪 run)

def preprocess\_text(text):

    words = word\_tokenize(text.lower())

    table = str.maketrans('', '', string.punctuation)

    words = [word.translate(table) for word in words if word.isalpha()]

    stop\_words = set(stopwords.words('english'))

    words = [word for word in words if word not in stop\_words]

    lemmatized\_words = [lemmatizer.lemmatize(word) for word in words]

    #stemmed\_words = [stemmer.stem(word) for word in words]

    preprocessed\_text = ' '.join(lemmatized\_words)

    return preprocessed\_text

Ta proces je šele polovica potrebnega predprocesiranja. Druga polovica pride v obliki vektorizacije. Pred tem uteženo združiva očiščene naslove in opise v en string. Z uporabo train\_test\_split funkcije razdeliva primere na učno in testno množico (razmerje 8:2).

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=8678686)

Množici X\_train in X\_test moramo spremeniti v vektorje. Za to obstaja več metod, midva sva se odločila za TF-IDF.

vectorizer = TfidfVectorizer()

X\_train\_vec = vectorizer.fit\_transform(X\_train[['clean\_head,clean\_desc']].values.astype('U'))

X\_test\_vec = vectorizer.transform(X\_test[['clean\_head,clean\_desc']].values.astype('U'))

Nato lahko pričnemo z učenjem.

## Modeli

Kot prvo sva se odločila preizkusiti nekaj različnih modelov za klasifikacijo in jih primerjati med sabo. Odločila sva se za tri osnovne in nekaj ''ensemble'' modelov. Sprva sva jih pognala s privzetimi parametri.

Uporabila sva naslednje modele:

* Odločitveno drevo. Pri tem je najpomembnejša globina drevesa, pri kateri se ustavimo graditi.
* Naivni Bayes
* kNN
* Hard voting
* Soft voting
* Weighted voting
* Bagging
* Naključni gozdovi
* Logistična regresija
* Adaboost
* XGboost

Tri najbolj točne modele uporabimo za nadaljnjo analizo. Sprva prikažemo poročilo klasifikacije za posamezne kategorije.

Najprej jih pošljemo v proces fine tuninga njihovih parametrov.

## Rezultati

Za analizo smo izbrali naključnih 15.000 primerov. Za vsakega izmed modelov sva izračunala točnost, ki sva jo na koncu prikazala s histogramom.

A bar graph with red and green bars

Description automatically generated

Točnosti

|  |  |
| --- | --- |
| **MODEL** | **TOČNOST** |
| Odločitveno drevo |  |
| Naivni Bayes |  |
| kNN |  |
| Hard voting |  |
| Soft voting |  |
| Weighted voting |  |
| Bagging |  |
| Naključni gozdovi |  |
| Logistična regresija |  |
| Adaboost |  |
| XGboost |  |

Pričakovano so se bolje odrezali ensemble modeli, in sicer Hard voting, soft voting in naključni gozdovi.

## Diskusija