# 2. Seminarska naloga: poročilo

# Inteligentni sistemi

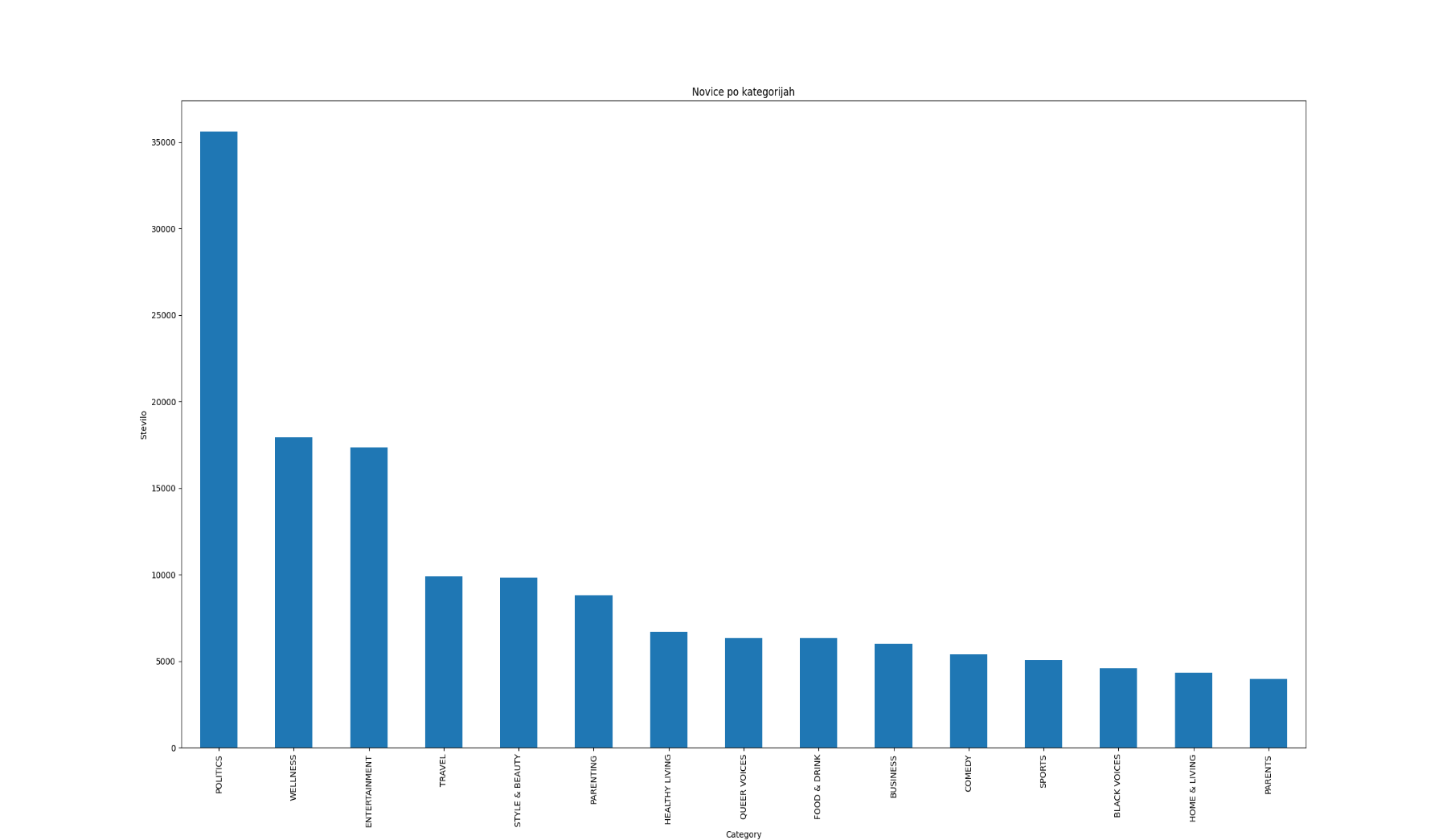
## 14. 1. 2024

## 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

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.

Novice so razdeljene v 15 kategorij, razporejene kot sledeče. Največ vnosov imajo kategorije POLITICS (35602), WELLNESS (17945) in ENTERTAINMENT (17362), najmanj pa PARENTS (3955).

Razporeditev vseh novic po kategorijah

Vidimo lahko, da so podatki zelo neuravnoteženi. Najbolj pogost razred POLITICS predstavlja kar 24% vseh primerov, najpogostejše tri skupaj pa kar 47.87%.

category\_counts = df['category'].value\_counts()

print(sum(category\_counts[0:3])/len(df))

Zato lahko predvidevamo, da se bodo rezultati raziskave precej razlikovali glede na podmnožico, ki jo bomo izbrali za obravnavo.

Za našo analizo smo se odločili izbrati 3 podmnožice podatkov: TODO

* Prvih 15.000 novic
* Uravnovešena podmnožica, kjer ima vsaka od 15 kategorij po 1000 novic
* 15.000 novic uravnoteženih s SMOTE

## Predpriprava besedila

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'))

TF-IDF je metoda, ki deluje na principu frekvence pojavitve določene besede v tekstu. TF del meri, kako pogosto se beseda pojavi v dokumentu, IDF del pa meri pomembnost besede glede na zbirko dokumentov. Besede, ki se pojavljajo v velikem številu dokumentov so kaznovane, saj to pomeni, da nosijo bolj splošen pomen in za klasifikacijo dokumenta niso toliko vredne. TF-IDF je metoda, ki združuje ta dva koncepta in da večjo težo besedam, ki so pogoste v specifičnem dokumentu, a redke v zbirki.

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. Ta model nima veliko parametrov, le alpha (parameter glajenja) in fit\_prior, s katerim določimo, če uporabljamo apriorno znanje verjetnosti iz osnovnih podatkov.
* kNN. Glavni parameter je k.
* Hard voting
* Soft voting
* Bagging
* Naključni gozdovi
* Logistična regresija
* Adaboost
* XGboost

Primer kode za naključne gozdove v začetni konfiguraciji:

# Random Forest

model\_rf = RandomForestClassifier(random\_state=12345)

# Start the timer

start\_time = time.time()

model\_rf.fit(X\_train\_vec, y\_train)

pred\_rf = model\_rf.predict(X\_test\_vec)

result\_rf = accuracy\_score(y\_test, pred\_rf)

print("Random Forest Accuracy:", result\_rf)

# Stop the timer

end\_time = time.time()

# Calculate the elapsed time

elapsed\_time = end\_time - start\_time

print(f"\tTime taken: {elapsed\_time} seconds")

print(classification\_report(y\_test, pred\_rf, zero\_division=1))

## Rezultati

Na treh prej omenjenih podmnožicah smo pognali vse modele. Dobili smo naslednje točnosti:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **MODEL** | **Točnost uravnovešena** | **Točnost neuravnovešenih** | **Točnost SMOTE neuravnovešena** | **Točnost prvih 15.000** | **Povprečje** |
| Odločitveno drevo | 0.5077 | 0.462 | 0.4323 | 0.709 | 0.53 |
| Naivni Bayes | 0.6917 | 0.4613 | 0.621 | 0.7833 | 0.6393 |
| kNN | 0.5683 | 0.1146 | 0.263 | 0.768 | 0.4285 |
| Hard voting | 0.638 | 0.6027 | 0.5687 | 0.786 | 0.6489 |
| Soft voting | 0.6223 | 0.597 | 0.5747 | 0.7853 | 0.6448 |
| Bagging | 0.5523 | 0.5326 | 0.4893 | 0.7577 | 0.583 |
| Naključni gozdovi | 0.6153 | 0.5963 | 0.545 | 0.7657 | 0.6306 |
| Logistična regresija | 0.6833 | 0.6427 | 0.634 | 0.7737 | 0.6834 |
| Adaboost | 0.513 | 0.3973 | 0.403 | 0.7043 | 0.5044 |
| XGboost | 0.6403 | 0.609 | / | 0.7703 | 0.6732 |

Zatem sva se lotila dodatne izboljšave modelov s pomočjo grid searcha. Za vsakega od modelov (razen votinga, ki je sestavljen iz večih modelov) ustvariva svojo mrežo parametrov, na katerih se požene algoritem. Vrne nam kombinacijo parametrov, ki privede do najboljših rezultatov.

Primer grid searcha za naključne gozdove:

param\_grid = {

  'n\_estimators': [100, 200, 300],

  'max\_depth': [None, 10, 20, 30],

  'min\_samples\_split': [2, 5, 10],

  'min\_samples\_leaf': [1, 2, 4],

  'max\_features': ['auto', 'sqrt']

}

# Create a base model

rf = RandomForestClassifier()

grid\_search = GridSearchCV(estimator=rf, param\_grid=param\_grid, cv=5)

grid\_search.fit(X\_train\_vec, y\_train)

best\_params = grid\_search.best\_params\_

print("Best parameters found: ", best\_params)

🡪Best parameters found: {'max\_depth': None, 'max\_features': 'sqrt', 'min\_samples\_leaf': 2, 'min\_samples\_split': 2, 'n\_estimators': 300}

Ko sva zaključila z grid searchom sva vse modele ponovno pognala, tokrat samo za uravnovešeno podmnožico.

Rezultati so se vidno izboljšali.

|  |  |  |
| --- | --- | --- |
| **MODEL** | **Prejšnja točnost** | **Nova točnost** |
| Odločitveno drevo | 0.5077 | 0.462 |
| Naivni Bayes | 0.6917 | 0.4613 |
| kNN | 0.5683 | 0.1146 |
| Hard voting | 0.638 | 0.6027 |
| Soft voting | 0.6223 | 0.597 |
| Bagging | 0.5523 | 0.5326 |
| Naključni gozdovi | 0.6153 | 0.5963 |
| Logistična regresija | 0.6833 | 0.6427 |
| Adaboost | 0.513 | 0.3973 |
| XGboost | 0.6403 | 0.609 |

Primerjava:

|  |  |  |
| --- | --- | --- |
| **Model** | **Točnost brez tuninga** | **Točnost po tuningu** |
| Logistična regresija |  |  |
| Hard voting |  |  |
| Naključni gozdovi |  |  |

## Diskusija