2. Seminarska naloga: poročilo

Inteligentni sistemi

14. 1. 2024

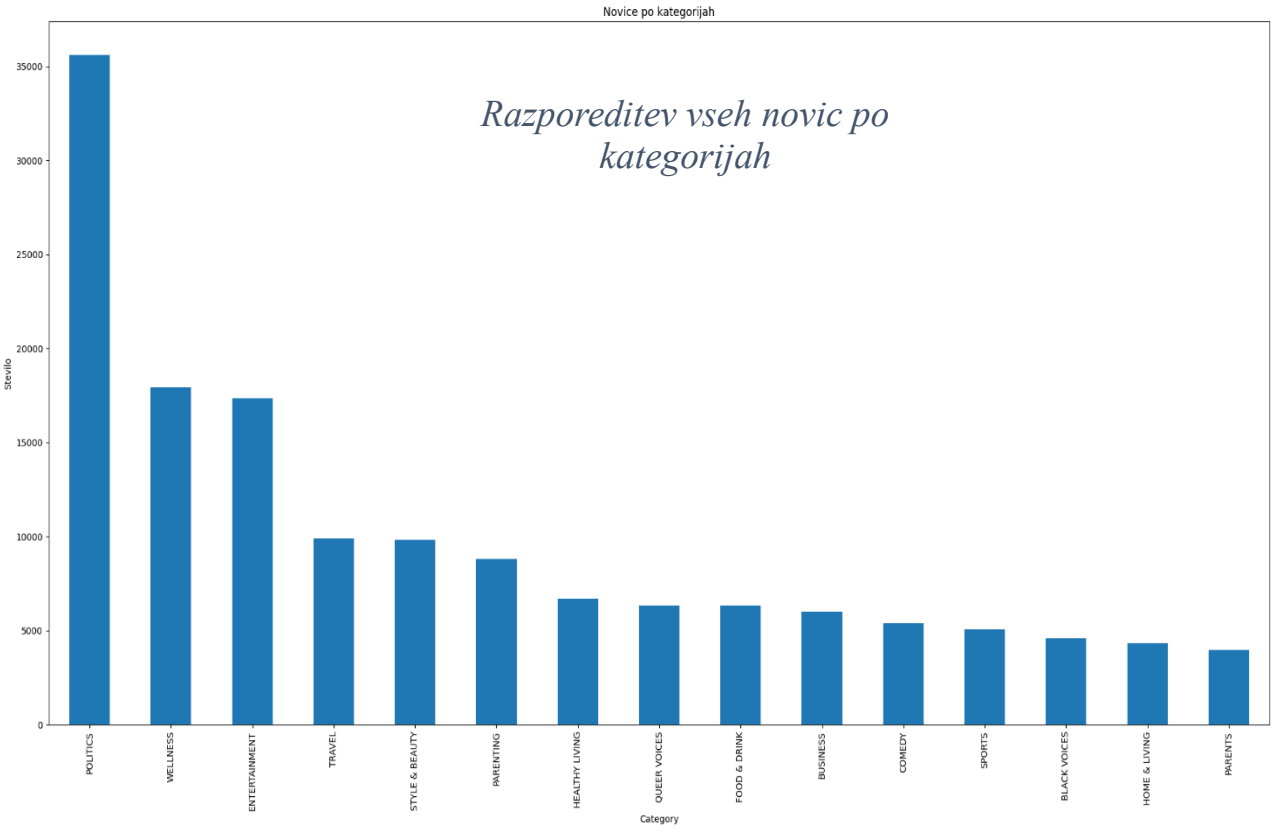
1. Uvod

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

2. Podatki

Podatki so podani v json obliki. Vsak vnos predstavlja podatke o eni novici. Vsega skupaj imamo 148122 novic. Za vsako novico je 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 prikazuje diagram spodaj. Največ vnosov imajo kategorije POLITICS (35602), WELLNESS (17945) in ENTERTAINMENT (17362), najmanj pa PARENTS (3955).



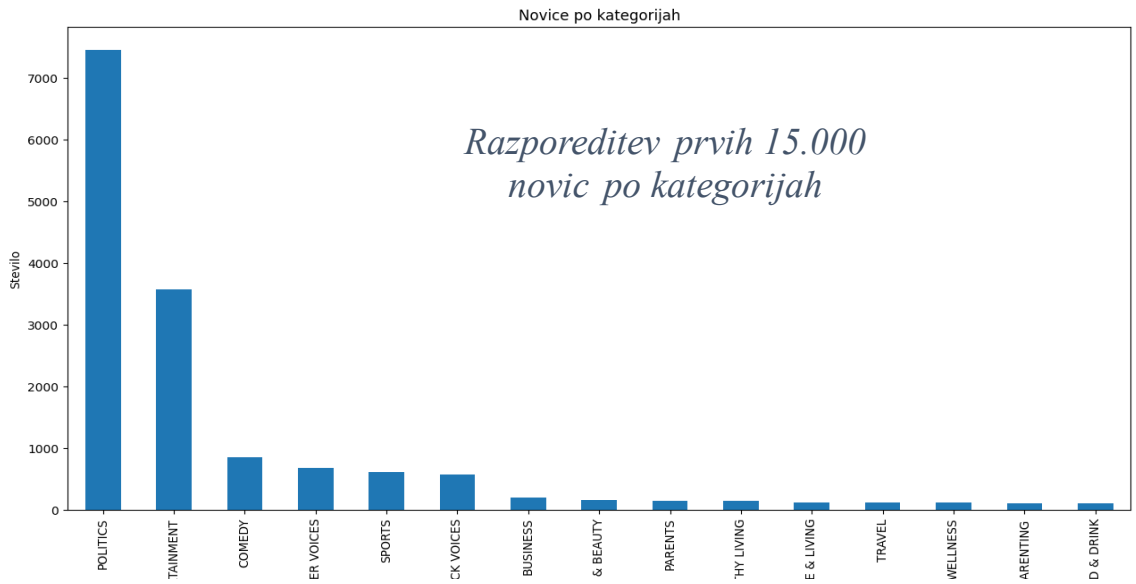
Vidimo lahko, da so podatki zelo neuravnoteženi. Najbolj pogost razred POLITICS predstavlja kar 24% vseh primerov, najpogostejši trije skupaj pa kar 47.87%.

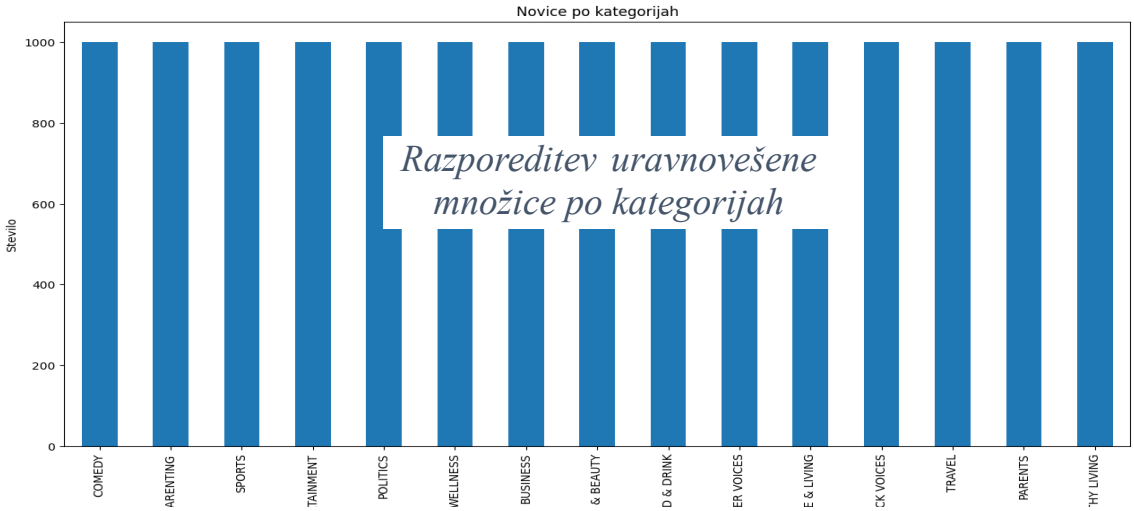
```
category_counts = df['category'].value_counts()
print(sum(category_counts[0:3])/len(df))
```

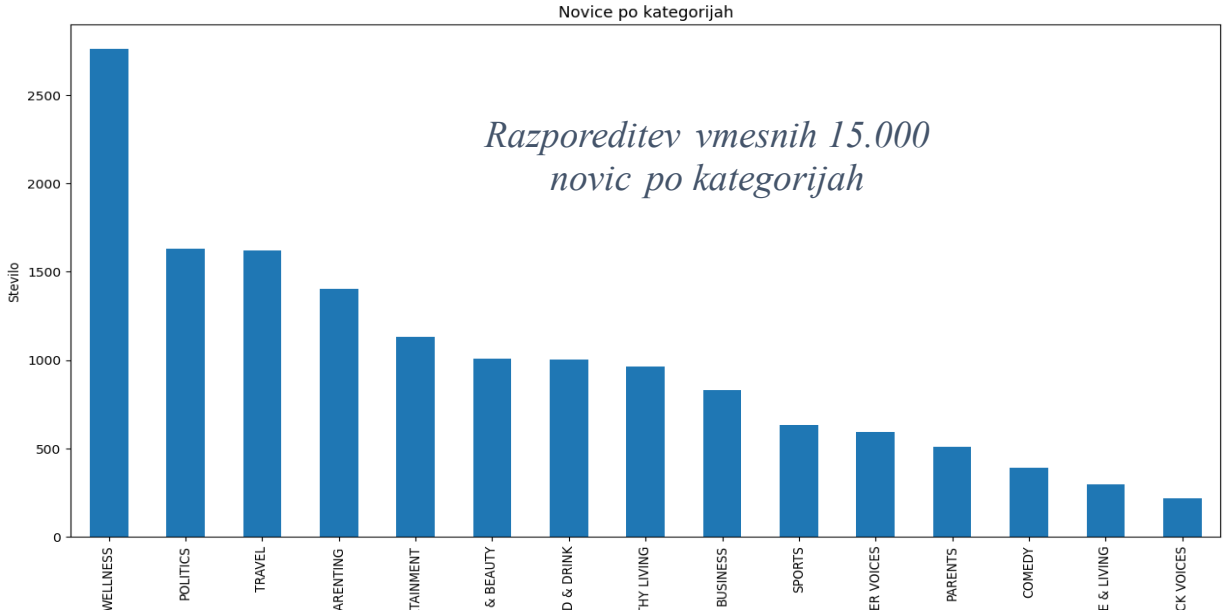
Zato lahko pričakujemo, da se bodo rezultati učenja precej razlikovali glede na podmnožico, ki jo bomo izbrali za obravnavo.

Za začetek našega učenja smo se odločili izbrati 4 podmnožice podatkov:

- Prvih 15.000 novic (zelo neuravnovešena množica)
- Uravnovešena podmnožica, kjer ima vsaka od 15 kategorij po 1000 novic
- 15.000 novic iz sredine podatkov (neuravnovešena množica)
- 15.000 novic uravnoteženih s SMOTE (tehnika, ki v neuravnoteženih učnih podatkih umetno ustvari primere z malo pojavitvami)







3. 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
```

```
new_data['clean_head'] = new_data['headline'].fillna('').apply(preprocess_text)
```

```
new_data['clean_desc'] = new_data['short_description'].fillna('').apply(preprocess_text)
```

Ta proces je šele polovica potrebnega predprocesiranja. Druga polovica pride v obliki vektorizacije. Pred tem uteženo združiva očiščene naslove in opise v en niz.

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=42)
```

Množici X_train in X_test moramo nato 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_text'].values.astype('U'))
X_test_vec = vectorizer.transform(X_test['clean_text].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.

4. Modeli

Kot prvo sva se na vseh podmnožicah odločila preizkusiti nekaj različnih modelov za klasifikacijo in jih primerjati med sabo. Odločila sva se za štiri 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, ki pove koliko sosedov iščemo vsakemu primeru.
- Hard voting (sprva sva uporabila bagging, linearno regresijo in naključne gozdove za estimatorje)
- Soft voting (sprva sva uporabila bagging, linearno regresijo in naivnega naključne gozdove za estimatorje)
- Bagging
- Naključni gozdovi. Pomembno je število dreves, ki jih uporabimo in njihova globina.
- 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))
```

Vsakemu klasifikatorju sva merila tudi čas in nato izpisala poročilo, v kateri je za vsako kategorijo podana preciznost, priklic, mera F1 in podpora.

5. 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	0.5278
Naivni Bayes	0.6917	0.4613	0.621	0.686	0.615
kNN	0.5683	0.1146	0.263	0.751	0.4242
Hard voting	0.638	0.6027	0.5687	0.782	0.6479
Soft voting	0.6223	0.597	0.5747	0.78	0.6435
Bagging	0.5523	0.5326	0.4893	0.7517	0.5815
Naključni gozdovi	0.6153	0.5963	0.545	0.7713	0.632
Logistična regresija	0.6833	0.6427	0.634	0.7833	0.6858
Adaboost	0.513	0.3973	0.403	0.7043	0.5044
XGboost	0.6403	0.609	/	0.7703	0.6732
Povprečje	0.60322	0.50155	0.503444	0.74889	0.59353

Točnost ni nujno dobra mera, še posebej v neuravnovešenih podatkih. Za to uporabimo raje mero F1.

Najbolj neuravnovešeni izbrani podatki so prvih 15.000. F1 vrednosti zanje so sledeče.

MODEL	Točnost prvih 15.000	F1 prvih 15.000
Odločitveno drevo	0.709	0.70
Naivni Bayes	0.686	0.59
kNN	0.751	0.74
Hard voting	0.782	0.76
Soft voting	0.78	0.77
Bagging	0.7517	0.74
Naključni gozdovi	0.7713	0.74
Logistična regresija	0.7833	0.75
Adaboost	0.7043	0.70
XGboost	0.7703	0.76

Zatem sva se lotila dodatne izboljšave modelov s pomočjo grid searcha. Za vsakega od 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:

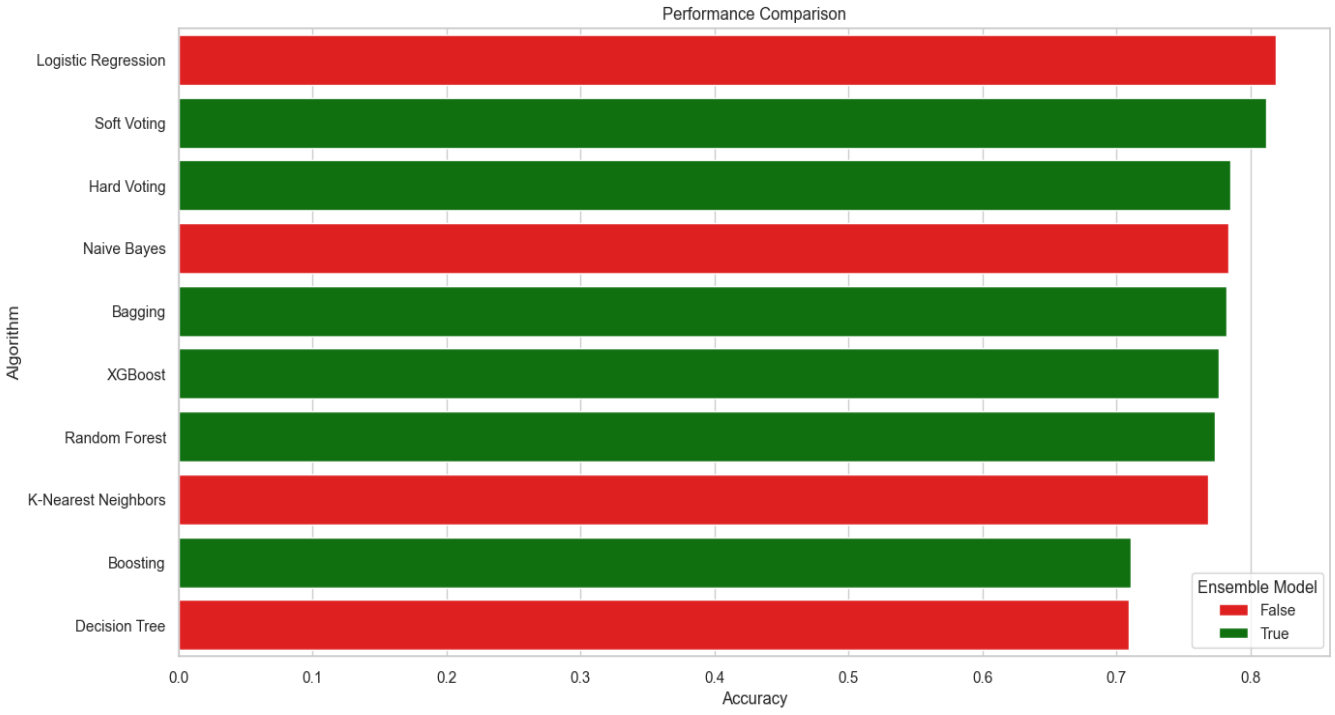
```
param_grid = {
  'n_estimators': [100, 200, 300],
  'max_depth': [None, 10, 20, 30],
  'min_samples_split': [2, 5, 10],
  'min_samples_leaf': [1, 2, 4],
  'max_features': ['auto', 'sqrt']
}

# 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 prvih 15.000 primerov. V parametre votinga sva zamenjala modele, za nove tri najboljše.

Rezultati so se vidno izboljšali.

MODEL	Prejšnja točnost	Nova točnost	Razlika
Odločitveno drevo	0.709	/	/
Naivni Bayes	0.686	0.7833	0.0973
kNN	0.751	0.768	0.017
Hard voting	0.782	0.7847	0.0027
Soft voting	0.78	0.8113	0.0311
Bagging	0.7517	0.7823	0.0306
Naključni gozdovi	0.7713	0.7733	0.002
Logistična regresija	0.7833	0.8187	0.0354
Adaboost	0.7043	0.7107	0.0064
XGboost	0.7703	0.776	0.0057



Izboljšanje je vidno pri vseh modelih. Največje izboljšanje je imel naivni Bayes s parametri:

```
model_NB = MultinomialNB(alpha=0.6, fit_prior=False)
```

Vidna izboljšava je bila še pri soft votingu in logistični regresiji, ki je bila tudi najboljši model.

Na logistični regresiji sva izvedela še cross-validation in dobila naslednje vrednosti:

Cross-Validation Scores: [0.81625 0.8125 0.80541667 0.805 0.81208333]

Kot zadnji korak sva se odločila izvesti logistično regresijo, ki se je izkazala za najboljši model, na celotnih podatkih.

Logistična regresija je bila izvedena z naslednjimi parametri:

```
model_lr = LogisticRegression(class_weight='balanced', C=10, penalty='l2',
max_iter=1000, solver='liblinear', random_state=42)
```

Dobila sva sledeče rezultate.

```
Logistic Regression Accuracy: 0.7277637130801687
        Time taken: 64.37095522880554 seconds
                precision    recall  f1-score   support

  BLACK VOICES       0.51      0.57      0.54       843
      BUSINESS       0.56      0.65      0.60      1190
        COMEDY       0.52      0.54      0.53      1100
 ENTERTAINMENT       0.77      0.73      0.75      3456
  FOOD & DRINK       0.75      0.80      0.78      1331
HEALTHY LIVING       0.33      0.33      0.33      1361
 HOME & LIVING       0.76      0.79      0.77       837
     PARENTING       0.62      0.60      0.61      1841
       PARENTS       0.37      0.36      0.37       787
      POLITICS       0.89      0.85      0.87      7070
  QUEER VOICES       0.76      0.73      0.74      1262
        SPORTS       0.75      0.82      0.79       983
STYLE & BEAUTY       0.84      0.86      0.85      1960
        TRAVEL       0.82      0.83      0.82      1956
      WELLNESS       0.70      0.70      0.70      3648

      accuracy                           0.73     29625
     macro avg       0.66      0.68      0.67     29625
  weighted avg       0.73      0.73      0.73     29625
```

Opazimo lahko, da je mera F1 na splošno slabša za manj pogoste razrede PARENTS in HEALTHY LIVING, kar se tudi sklada z našimi pričakovanji. Podobno velja tudi za vrednosti preciznosti in priklica.

Izvedemo še cross validation:

```
 -> Cross-Validation Scores: [0.72607595 0.72987342 0.72405063 0.72548523
0.72725738 0.72101266
 0.72270042 0.73162292 0.72470251 0.72166428]
```

Naš model se izvede s podobno natančnostjo na vseh podmnožicah pri cross validationu, kar kaže, da je naš model konsistenten, ne glede na delitev podatkov.

6. Transformerji (nevronske mreže)

Odločila sva se tudi za poskus s transformerji. Po primeru iz laboratorijskih vaj sva zgradila mrežo in na njej pognala prvih 15.000 primerov. Mreža ima 4 skrite sloje z ReLU aktivacijsko funkcijo. Zadnji sloj uporablja softmax funkcijo, ki se uporablja pri večrazredni klasifikaciji. Določila sva, da se mreža uči 40 epochov.

```
embed_dim = 64  # Embedding size for each token
num_heads = 4  # Number of attention heads
ff_dim = 64  # Hidden layer size in feed forward network inside transformer

inputs = layers.Input(shape=(maxlen,))
#437
embedding_layer = TokenAndPositionEmbedding(maxlen, len, embed_dim)
x = embedding_layer(inputs)
transformer_block = TransformerBlock(embed_dim, num_heads, ff_dim)
x = transformer_block(x)
x = layers.GlobalMaxPooling1D()(x)
x = layers.Dropout(0.1)(x)
x = layers.Dense(16*16, activation="relu")(x)
x = layers.Dropout(0.1)(x)
x = layers.Dense(16*32, activation="relu")(x)
x = layers.Dropout(0.1)(x)
x = layers.Dense(16*16, activation="relu")(x)
x = layers.Dropout(0.1)(x)
x = layers.Dense(16*8, activation="relu")(x)
x = layers.Dropout(0.1)(x)
outputs = layers.Dense(16, activation="softmax")(x)

model = keras.Model(inputs=inputs, outputs=outputs)

model.compile("adam", "sparse_categorical_crossentropy", metrics=["accuracy"])
history = model.fit(
    x_train, y_train, batch_size=64, epochs=40, validation_data=(x_val, y_val)
)
```

Med izvajanjem učenja sva lahko opazila, da se je do neke točke točnost nad testno množico zviševala, nato pa se je zaradi overfittanja pričela nižati, medtem ko je točnost nad učno množico nadaljevala z rastjo.

Vrednosti v prvem epochu:

```
Epoch 1/40
176/176 [==============================] - 76s 416ms/step - loss: 1.5746 -
accuracy: 0.5249 - val_loss: 1.4773 - val_accuracy: 0.5771
```

Epoch z najvišjo točnostjo val_accuracy:

```
Epoch 10/40
176/176 [==============================] - 73s 413ms/step - loss: 1.0577 -
accuracy: 0.6900 - val_loss: 1.1905 - val_accuracy: 0.6469
```

Zadnji epoch:

```
Epoch 40/40
176/176 [==============================] - 65s 371ms/step - loss: 0.3718 -
accuracy: 0.8777 - val_loss: 2.0354 - val_accuracy: 0.6056
```

Model je imel manjšo točnost kot vsi ostali klasifikatorji, kar pomeni, da ni bil optimalno zgrajen in bi potreboval spremembe, predvsem v skritih slojih, a se v to nisva globlje spuščala.

7. Zaključek

V najini nalogi je ostalo še veliko prostora za izboljšave. Glavna bi seveda bila, da bi iz spleta pobrala celotne članke, a to porabi zelo veliko časa. In čas je na splošno tudi glavna ovira pri celotni nalogi. Še posebej pri grid searchu pri iskanju hiperparametrov, kjer lahko preveliko zastavljeno iskalno okolje porabi tudi več ur, da najde prave parametere in še ti so lahko na koncu nepravi. Predvsem večje število možnih parametrov bi lahko prineslo boljše rezultate in tu je čas velik problem za izboljšavo.

Na splošno pa nisva srečala nobenih presenetljivih rezultatov, saj so se ti skladali z našimi pričakovanji baziranimi na najinemu znanju strojnega učenja.