IRTM_Project_TudorAndrei_Dumitrascu

January 23, 2022

0.1 IRTM Project 2

Dumitrascu Tudor Andrei

[ ]:

```
import spacy
import pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegressionCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from tqdm import tqdm
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.metrics import classification_report, f1_score
from statistics import mean
import numpy as np
```

[ ]:

```
# !python -m spacy download en_core_web_sm
```

[ ]:

```
nlp = spacy.load("en_core_web_sm", exclude=['ner', 'lemmatizer'])
```

1 Load data

[ ]:

```
data = pd.read_csv("Lyrics-Genre-Train.csv").drop(['Song', 'Song year',␣
 ↪'Artist', 'Track_id'], axis=1)
data_test = pd.read_csv("Lyrics-Genre-Test-GroundTruth.csv").drop(['Song',␣
 ↪'Song year', 'Artist', 'Track_id'], axis=1)
```

[ ]:

```
def generate_pos(x) -> str:
    doc = nlp(x)
    results = []
    for i, token in enumerate(doc):
        results.append(token.pos_)

    return " ".join(results)
```

```
[ ]: data['POS'] = data['Lyrics'].apply(generate_pos)
     data_test['POS'] = data_test['Lyrics'].apply(generate_pos)
```

```
[ ]: def train_eval(model, X_train, y_train, X_test, y_test, encoder):
         print(str(model))
         model.fit(X_train, y_train)
         print("Train")
         y_hat = model.predict(X_train)
         print(f1_score(y_train, y_hat, average='weighted'))
         print("Test")
         y_hat = model.predict(X_test)
         print(f1_score(y_test, y_hat, average='weighted'))
```

2 Implementation

2.1 Verse Features

```
[ ]: def pos_count(x, pos):
         return sum([y == pos for y in x.split() ])

     # count the part of speech tags in the whole song
```

```
[ ]: poeses = set(data.loc[0,'POS'].split())
     poeses
```

```
[ ]: {'ADJ',
      'ADP',
      'ADV',
      'AUX',
      'CCONJ',
      'DET',
      'NOUN',
      'PRON',
      'PROPN',
      'PUNCT',
      'SPACE',
      'VERB'}
```

```
[ ]: for pos in tqdm(poeses):
         data[ pos + "_count"] = data['POS'].apply(lambda x: pos_count(x, pos))
         data_test[ pos + "_count"] = data_test['POS'].apply(lambda x: pos_count(x,␣
      ↪pos))
```

```
     100%|     | 12/12 [00:07<00:00,  1.52it/s]
```

```
[ ]: # Count the number of verses
     data['newln_count'] = data['Lyrics'].apply(lambda x: x.count("\n"))
     data_test['newln_count'] = data_test['Lyrics'].apply(lambda x: x.count("\n"))
```

```
# Count the avg verse length
data['mean_verse_len'] = data['Lyrics'].apply(lambda x: mean([len(y) for y in x.
 ↪split("\n")]))
data_test['mean_verse_len'] = data_test['Lyrics'].apply(lambda x: mean([len(y)␣
 ↪for y in x.split("\n")]))
# count the avg no. of words per verse
data['mean_word_count_per_verse']  = data['Lyrics'].apply(lambda x: mean([len(y.
 ↪split()) for y in x.split("\n")]))
data_test['mean_word_count_per_verse']  = data_test['Lyrics'].apply(lambda x:␣
 ↪mean([len(y.split()) for y in x.split("\n")]))
```

```
[ ]: X_train = data.drop(['Genre', 'Lyrics', 'POS'], axis=1)
     X_test = data_test.drop(['Genre', 'Lyrics', 'POS'], axis=1)
     enc = LabelEncoder()
     y_train = enc.fit_transform(data['Genre'])
     y_test = enc.transform(data_test['Genre'])
```

```
[ ]: train_eval(SVC(class_weight='balanced'), X_train, y_train, X_test, y_test, enc)
```

```
SVC(class_weight='balanced')
Train
0.2917416059978742
Test
0.277862611528458
```

```
[ ]: train_eval(DecisionTreeClassifier(class_weight="balanced", criterion='entropy',␣
      ↪max_depth=10), X_train, y_train, X_test, y_test, enc)
```

```
DecisionTreeClassifier(class_weight='balanced', criterion='entropy',
                       max_depth=10)
Train
0.3493602597226395
Test
0.23033025647020453
```

```
[ ]: train_eval(RandomForestClassifier(n_jobs=-1), X_train, y_train, X_test, y_test,␣
      ↪enc)
```

```
RandomForestClassifier(n_jobs=-1)
Train
0.9998919615100562
Test
0.29142742830782753
```

2.2 TFIDF

[ ]:
```
tfidf =  TfidfVectorizer(ngram_range=(1,3), max_features=2000, max_df=0.8,␣
 ↪min_df=0.2)
X_train = tfidf.fit_transform(data['Lyrics'])
X_test = tfidf.transform(data_test['Lyrics'])
```

[ ]:
```
enc = LabelEncoder()
y_train = enc.fit_transform(data['Genre'])
y_test = enc.transform(data_test['Genre'])
```

[ ]:
```
train_eval(SVC(class_weight='balanced'), X_train, y_train, X_test, y_test, enc)
```

```
SVC(class_weight='balanced')
Train
0.5939188926029967
Test
0.30550260205341334
```

[ ]:
```
train_eval(RandomForestClassifier(n_jobs=-1), X_train, y_train, X_test, y_test,␣
 ↪enc)
```

```
RandomForestClassifier(n_jobs=-1)
Train
0.9992441696631217
Test
0.295679015623803
```

2.3 Part of speech tag

[ ]:
```
tfidf =  TfidfVectorizer(ngram_range=(1,3), max_features=300)
X_train = tfidf.fit_transform(data['POS'])
X_test = tfidf.transform(data_test['POS'])
```

[ ]:
```
enc = LabelEncoder()
y_train = enc.fit_transform(data['Genre'])
y_test = enc.transform(data_test['Genre'])
```

[ ]:
```
train_eval(SVC(class_weight='balanced'), X_train, y_train, X_test, y_test, enc)
```

```
SVC(class_weight='balanced')
Train
0.4188555378773796
Test
0.3203567772402991
```

[ ]:
```
train_eval(RandomForestClassifier(n_jobs=-1), X_train, y_train, X_test, y_test,␣
 ↪enc)
```

```
RandomForestClassifier(n_jobs=-1)
Train
0.9998379632304489
Test
0.30130022882875435
```

2.4 Word len

```
[ ]: enc = LabelEncoder()
     y_train = enc.fit_transform(data['Genre'])
     y_test = enc.transform(data_test['Genre'])
```

```
[ ]: def word_len(string, max_pad=200):
         doc = nlp.tokenizer(string)
         results = []
         for i, token in enumerate(doc):
             results.append(len(token))
         results = np.asarray(results)
         if len(results) < max_pad:
             results = np.pad(results, (0, max_pad - len(results)))
         else:
             results = results[:200]
         return results
```

```
[ ]: X_train = data['Lyrics'].apply(word_len)
     X_train = np.stack(X_train.values)
```

```
[ ]: X_test = data_test['Lyrics'].apply(word_len)
     X_test = np.stack(X_test.values)
```

```
[ ]: min_max = MinMaxScaler().fit(X_train)
     X_train = min_max.transform(X_train)
     X_test = min_max.transform(X_test)
```

```
[ ]: train_eval(SVC(class_weight='balanced'), X_train, y_train, X_test, y_test, enc)
```

```
SVC(class_weight='balanced')
Train
0.5391034891259431
Test
0.1806859460442372
```

```
[ ]: train_eval(RandomForestClassifier(n_jobs=-1), X_train, y_train, X_test, y_test,␣
      ↪enc)
```

```
RandomForestClassifier(n_jobs=-1)
Train
0.9998919615100562
```

```
Test
0.16504548553924356
```

[ ]:

```
```