Progetto Programmazione Applicazioni Data Intensive A.A. 2019/2020

Autore: ivan.perazzini@studio.unibo.it

NOTA: la versione del progeto in formato HTML e PDF sono state editate per eliminare o ridurre gli output di particolare lunghezza.

Questo progetto mira a costruire un classificatore che riconosca il genere di una canzone a partire dal suo testo. Si tratta quindi di un problema supervisionato, con variabili di input non strutturate e l'output che si deve prevedere è di tipo discreto.

I generi presi in esame saranno: pop, hip hop, rap, rock, r&b.

I dataset utilizzati sono stati scaricati da Kaggle:

-https://www.kaggle.com/danield2255/data-on-songs-from-billboard-19992019
-https://www.kaggle.com/neisse/scrapped-lyrics-from-6-genres
-https://www.kaggle.com/imuhammad/audio-features-and-lyrics-of-spotify-songs

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import nltk
import random
import re
import seaborn as sns
from sklearn.dummy import DummyClassifier
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score, f1_score
from nltk import sent_tokenize, word_tokenize
#from sklearn.svm import SVC

%matplotlib inline

DATASET1_NAME = "spotify_songs.csv"
DATASET2_NAME = "lyrics-data.csv"
DATASET3_NAME = "artists-data.csv"
DATASET4_NAME = "billboardHot100_1999-2019.csv"
```

In [2]:

```
import os.path
if (not os.path.exists(DATASET1_NAME) or not os.path.exists(DATASET2_NAME) or not os.path.exists(DA
TASET3_NAME)):
    raise FileNotFoundError("ERROR: One or more datasets missing!")
```

In [3]:

```
#pd.set_option('display.max_columns', 500)
spot_columns = ["lyrics", "language", "playlist_genre"]
spot_types = {"lyrics": "string", "language":"category", "playlist_genre":"string"}
df_spot = pd.read_csv("spotify_songs.csv", usecols=spot_columns, dtype=spot_types)

data2_columns = ["Lyric", "Idiom", "ALink"]
data3_columns = ["Genre", "Link"]
data2_types = {"Lyric": "string", "Idiom":"category"}
data3_types = {"Genre": "string"}
df_2 = pd.merge(pd.read_csv(DATASET2_NAME, usecols=data2_columns, dtype=data2_types), pd.read_csv(D
ATASET3_NAME, usecols=data3_columns, dtype=data3_types), left_on="ALink", right_on="Link")

data4_columns = ["Lyrics", "Genre"]
```

```
data4_types = {"Lyrics": "string", "Genre": "string"}
df_billboard = pd.read_csv(DATASET4_NAME, usecols=data4_columns, dtype=data4_types)
```

Due dataset contengono anche una colonna che indica la lingua: in questa istanza analizzeremo testi principalmente in Inglese, ma è molto probabile che qualche parola in altre lingue (es: Spagnolo) o espressioni popolari compaiano fra i dati.

Selezioniamo quindi le colonne rilevanti nei dataset e uniamoli tutti tramite operazione di merge.

In [4]:

```
df_eng = df_spot[df_spot["language"] == "en"]
df2_eng = df_2[df_2["Idiom"] == "ENGLISH"]
```

In [5]:

```
df_eng = df_eng.loc[:, ["lyrics", "playlist_genre"]]
df2_eng = df2_eng.loc[:, ["Lyric", "Genre"]]

df_eng.rename(columns={"playlist_genre":"genre"}, inplace=True)
df2_eng.rename(columns={"Lyric":"lyrics", "Genre":"genre"}, inplace=True)
df_billboard.rename(columns={"Lyrics":"lyrics", "Genre":"genre"}, inplace=True)
```

In [6]:

```
df = df_eng.append(df2_eng)
df = df.append(df_billboard)
del df_eng
del df2_eng
del df_spot
del df_2
del df_billboard
```

Come si può vedere, già ad una prima analisi risultano essere presenti molti duplicati, si effettuerà quindi una prima semplice scrematura eliminando le istanze con testo duplicato

In [7]:

```
print(df.shape)
print(df.nunique())
```

```
(236860, 2)
lyrics    107039
genre       2374
dtype: int64
```

In [8]:

```
df["lyrics"] = df["lyrics"].str.lower();
df["genre"] = df["genre"].str.lower();
df = df.drop_duplicates(subset="lyrics")
print(df.nunique())
```

```
lyrics    106982
genre       2371
dtype: int64
```

Si analizzano ora i generi presenti e i relativi numeri di canzoni associati.
Come si può vedere c'è un forte sbilanciamento in favore di canzoni catalogate come "rock", "pop" e "hip hop".

Si può anche notare però che alcuni generi sono in realtà insiemi di sottogeneri (es: "country, rock") oppure generi già presenti ma scritti in altro modo (es: "r&b"/"r&;b").

In [9]:

```
pd.set_option('display.max_rows', None)
df["genre"].value_counts()
```

Effettuiamo quindi l'accorpamento di canzoni catalogate in più generi a quelle catalogate in generi con numero rilevante di entry ed unifichiamo il modo in cui alcuni generi sono definiti. Gli accorpamenti verranno eseguiti in modo da aggiungere più entry possibili a generi potenzialmente rilevanti.

In particolare (in ordine di priorità):

- Tutte le canzoni che hanno fra i generi il country vengono accorpate al country.
- Tutte le canzoni che hanno fra i generi il latin verranno accorpate al genere latin. (A scopo di pulizia, poi verranno ignorate)
- Tutte le canzoni che hanno fra i generi i'edm verranno accorpate al genere edm. (A scopo di pulizia, poi verranno ignorate)

E così via. Dopodichè, tutte le rimanenti canzoni con più di un genere verranno catalogate con il primo dei generi presenti.

In [10]:

```
df["genre"] = [re.sub(".*country.*",'country',g) for g in df["genre"]]
df["genre"] = [re.sub(".*latin.*",'latin',g) for g in df["genre"]]
df["genre"] = [re.sub(".*edm.*",'edm',g) for g in df["genre"]]
df["genre"] = [re.sub(".*r&;?b.*",'r&b',g) for g in df["genre"]]
df["genre"] = [re.sub(".*rap.*",'rap',g) for g in df["genre"]]
df["genre"] = [re.sub(".*hip\ ?-?hop.*",'hip hop',g) for g in df["genre"]]
df["genre"] = [re.sub(".*pop.*",'pop',g) for g in df["genre"]]
df["genre"] = [re.sub(".*rock.*",'rock',g) for g in df["genre"]]

df["genre"] = [re.sub("(,|;).*",'',g) for g in df["genre"]]
```

In [11]:

```
df["genre"].value_counts()
```

Out[11]:

```
rock            50765
pop             30744
hip hop         13551
rap              4229
r&b              4053
edm              1464
country          1262
latin             820
sertanejo          41
samba              39
funk carioca       14
Name: genre, dtype: int64
```

Prima di ripulire i testi o bilanciare i numeri di entry manualmente, iniziamo ad eliminare testi con caratteristiche fuori dalla media, per esempio la lunghezza del testo.

Si analizzano quindi i testi particolarmente lunghi o corti per vedere se sia probabile che questi non contengano informazioni interessanti.

Risulta che i testi corti sono spesso o incompleti, o composti solo da indicazioni, spesso relative al fatto che la canzone non contiene testo.

In [12]:

```
pd.set_option('display.max_rows', 1000)

df_sorted = df.copy(deep=True)
df_sorted.index = df['lyrics'].str.len()
df_sorted = df_sorted.sort_index(ascending=False).reset_index(drop=True)
df_sorted["lyrics"].sort_index(ascending=False)[:20]
```

Out[12]:

```
106981               play nice
106980               no lyrics
106979               beautiful
106978            don't lyrics
106977            hallelujah

106976         improvised song
```

```
106975        all systems down
106974        the son. the sun
106973       instrumental song
106972       instrumental only
106971       kick it like this
106970       without your love
106969       (one, two, three)
106968      that beat is fire!
106967      who told you how?.
106966     smooth that out now
106965     (instrumental song)
106964     written by jay kay.
106963    (instrumental track)
106962    acapella, no lyrics.
Name: lyrics, dtype: string
```

I testi lunghi invece per la maggior parte sembrano contere informazioni utili, ma alcuni contengono per la maggior parte indicazioni su spartiti o simili.

Andando più a fondo nella ricerca si nota anche come spesso siano inserite nel testo annotazioni, per esempio "[laughs]" ad indicare che il cantante sta ridendo, oppure "(feat. -nomi-.)" e così via.

Si notano anche, oltre alla punteggiatura, altri caratteri non alfabetici.

Verranno utilizzate queste informazioni successivamente, in fase di pulizia dei testi tramite espressioni regolari.

In [13]:

```
df_sorted["lyrics"].sort_index(ascending=True)[:10]
```

Out[13]:

```
0    e|-----------------------------------------...
1    e-------5-7-----7-|-8-----8-2-----2-|-0-------...
2    notation legend.-. / = slide. = palm mute. b =...
3    words and music by f.iommi, t.butler, w.ward, ...
4    na workin' on a weekend like usual way off in ...
5    it's funny, the shit i put on this song ain't ...
6    yeah i'm home now, it's over now, so... yeah s...
7    na astro, yeah sun is down, freezin' cold that...
8    wheezy outta here hot, hot, hot, hot hot, hot,...
9    yo, sing this shit, are y'all fuckin' dumb? et...
Name: lyrics, dtype: string
```

In [14]:

```
del df_sorted
```

Diamo quindi un'occhiata alle lunghezze dei testi, per vedere se si può scremare facilmente parte del dataset.

Come prevedibile dalle analisi precedenti, si nota una grande differenza fra le lunghezze massime e minime, con conseguente deviazione importante.

In [15]:

```
lengths = df["lyrics"].str.len()
print(lengths.describe())
```

```
count    106982.000000
mean       1550.456301
std        1112.575856
min           9.000000
25%         889.000000
50%        1278.000000
75%        1882.000000
max       64047.000000
Name: lyrics, dtype: float64
```

La maggior parte dei testi sembra avere fra i 1000 e i 3500 caratteri.

Si provvederà quindi a rimuovere le entry fuori dall'intervallo

In [16]:

```
np.quantile(lengths, 0.5), np.quantile(lengths, 0.95)
```

Out[16]:

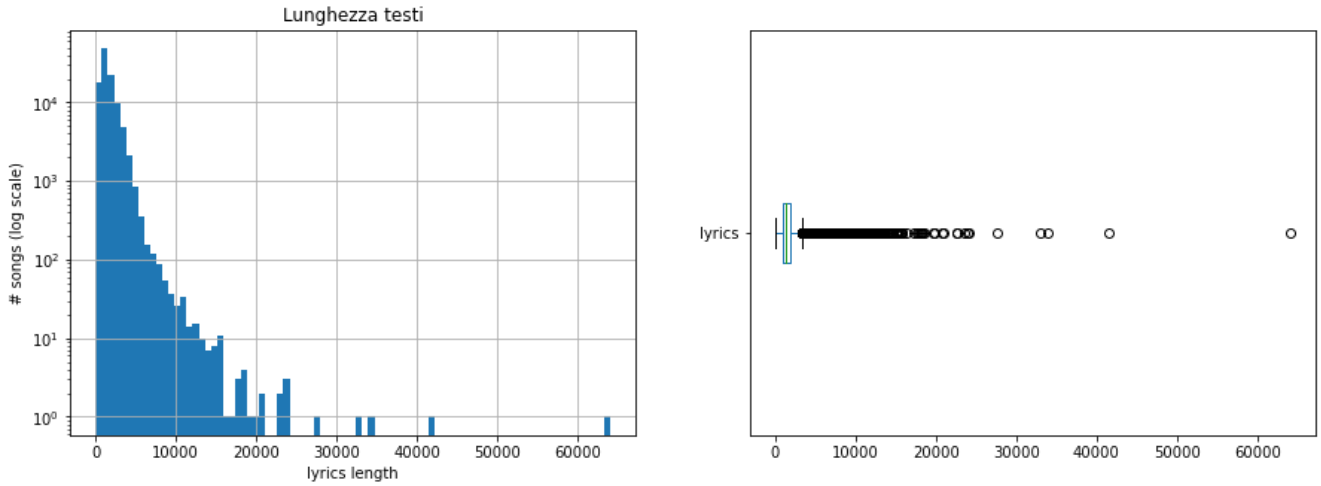
```
(1278.0, 3495.949999999997)
```

In [17]:

```
plt.figure(figsize=(15,5))

plt.subplot(121)
plt.title("Lunghezza testi")
plt.xlabel("lyrics length")
plt.ylabel("# songs (log scale)")
lengths.hist(bins=85, log=True);

plt.subplot(122)
lengths.plot.box(vert=False);
```



In [18]:

```
keep = lengths.between(1000, 3500, inclusive = True)
df = df[keep]
df.info()
del lengths
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 67045 entries, 2 to 97224
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   lyrics  67045 non-null  string
 1   genre   67045 non-null  object
dtypes: object(1), string(1)
memory usage: 1.0+ MB
```

Vediamo ora la nuova distribuzione di canzoni nei vari generi.

La maggior parte delle entry eliminate sembra appartenere alle classi già sovra rappresentate.

In [19]:

```
#Vediamo ora di nuovo come sono distribuiti i generi
df["genre"].value_counts()
```

Out[19]:

```
rock            25748
```

```
rock             25740
pop              22408
hip hop          10122
r&b               3299
rap               2767
country           1090
edm                932
latin              636
samba               18
sertanejo           17
funk carioca         8
Name: genre, dtype: int64
```

In [20]:

```
stop_words = nltk.corpus.stopwords.words("english")
```

Esploriamo ora i termini più comuni.

Senza stopwords, il risultato è abbastanza prevedibile, quindi le useremo.
Gli n-gram (sono stati testati bi e trigram) più comuni sembrano essere in larga parte ripetizioni di parole.

Andando più a fondo nella lista possiamo però trovare anche alcuni dei pattern e anomalie già notate analizzando i testi più lunghi.
Procediamo quindi alla pulizia del testo.

In [21]:

```
#from https://stackoverflow.com/questions/16078015/list-the-words-in-a-vocabulary-according-to-occurrence-in-a-text-corpus-with-sc
def get_popular_words(vect, vect_fit):
    sum_words = cv_fit.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in cv.vocabulary_.items()]
    return sorted(words_freq, key = lambda x: x[1], reverse=True)
```

In [22]:

```
cv = CountVectorizer(stop_words=stop_words)
cv_fit = cv.fit_transform(df["lyrics"])
#popular_words_100 = get_popular_words(cv, cv_fit)[:100]
get_popular_words(cv, cv_fit)[:10]
```

Out[22]:

```
[('like', 154748),
 ('know', 151615),
 ('love', 143927),
 ('oh', 138808),
 ('got', 115797),
 ('get', 111684),
 ('yeah', 100787),
 ('go', 91686),
 ('let', 85947),
 ('baby', 85610)]
```

Procedura di pulizia dei testi:

Viene analizzato il testo di ogni canzone: per ognuno, si eliminano preventivamente tutte le espressioni fra parentesi quadre (considerate tutte annotazioni), le espressioni "(laughs)" e "feat. ... .".

Dopodichè si divide il testo in frasi tramite funzione sent_tokenize di nltk e per ogni frase si cercano ed eliminano pattern relativi a crediti per musica e testi.

Infine si ricompongono i testi e si eliminano tutti i caratteri non appartenenti ad alfabeti occidentali comuni.

In [23]:

```
def clean_sentence(sent):
    pattern = ".*(music|lyrics) (by)?.*:.*?(?=\.|;)"
    return re.sub(pattern, '', sent)

def clean_text(text):
```

```
    patterns = ["\[.*\]", "\(\ *laughs\ *\)", "feat\..*?(?=\.)"]
    valid_chars = "[^a-z ]"
    for p in patterns:
        text = re.sub(p, '', text)
    text = sent_tokenize(text)
    for i, s in enumerate(text):
        text[i] = clean_sentence(s)
    text = ' '.join(text)
    text = re.sub(valid_chars, '', text)
    return text

df["lyrics"] = df["lyrics"].apply(lambda x: clean_text(x))
```

In seguito a questa pulizia alcuni testi sono stati drasticamente ridotti o completamente cancellati.

Si effettua un'operazione di scrematura basata sulle lunghezze dei testi in modo analogo a quanto già fatto all'inizio dell'analisi dei dati.

In [24]:

```
lengths = df["lyrics"].str.len()
print(lengths.describe())
np.quantile(lengths, 0.5), np.quantile(lengths, 0.95)
```

```
count    67045.000000
mean      1459.894355
std        632.332046
min          0.000000
25%       1090.000000
50%       1356.000000
75%       1779.000000
max       3405.000000
Name: lyrics, dtype: float64
```

Out[24]:

```
(1356.0, 2709.0)
```

In [25]:

```
keep = lengths.between(1300, 2700, inclusive = True)
df = df[keep]
```

In [26]:

```
del keep
del lengths
```

In [27]:

```
df["genre"].value_counts()
```

Out[27]:

```
pop             13100
rock            10418
hip hop          4151
r&b              2277
rap              1680
country           680
edm               577
latin             429
samba               7
sertanejo           6
funk carioca        4
Name: genre, dtype: int64
```

Verranno mantenuti solo brani appartenti ai generi elencati all'inizio.

Il country ha un numero di canzoni piuttosto basso, ma sperimentalmente è stato verificato che il genere è comunque facilmente

Il country ha un numero di canzoni piuttosto basso, ma sperimentalmente è stato verificato che il genere è comunque facilmente riconoscibile anche con un piccolo campione, quindi viene mantenuto.

Una volta finito di manipolare i generi, convertiamoli in valore categorico

In [28]:

```
df = df[~df["genre"].isin(["sertanejo", "funk carioca", "samba", "latin", "edm"])]
```

In [29]:

```
df = df.astype({"genre" : "category"})
```

Classificazione

Il modello di classificazione adottato sarà quello della Regressione Logistica: gli iperparametri migliori verranno cercati e valutati tramite grid search cross validation.

In [30]:

```
from nltk import stem, word_tokenize
stemmer = stem.PorterStemmer()
def tokenize_stem(words):
    words.lower()
    return [stemmer.stem(w) for w in word_tokenize(words) if w not in stop_words]
def tokenize(words):
    words.lower()
    return [w for w in word_tokenize(words) if w not in stop_words]
```

In [31]:

```
def draw_heat_map(pred, actual, classes):
    plt.figure(figsize=(12,12));
    cm = confusion_matrix(actual, pred, normalize="true")
    sns.heatmap(cm, square=True, annot=True, cmap="coolwarm",
    xticklabels=classes, yticklabels=classes)
    plt.xlabel("Predicted");
    plt.ylabel("Actual");
```

In [32]:

```
def get_weighted_words(model, tokenizer, ascending=False):
    return pd.DataFrame(model.coef_,
                        columns=tokenizer.get_feature_names()).T.sort_values(by=0,
ascending=ascending)
```

In [33]:

```
def some_by_genre(n, genre):
    gen_filtered = df[df["genre"] == genre]
    return gen_filtered.sample(n=n, random_state=12)
```

In [82]:

```
def compare_models(lh_score, rh_score, N1, N2):
    Z=2.58
    d = abs(lh_score - rh_score)
    sigma_d = np.sqrt(lh_score * (1-lh_score) / N1 + rh_score * (1-rh_score) / N2)
    interval_min, interval_max = d - Z * sigma_d, d + Z * sigma_d
    relevant = "rilevante" if interval_min * interval_max > 0 else "non rilevante"
    return f"La differenza fra i due modelli (confidenza 99%) è statisticamente
{relevant}.\n\tIntervallo: [{interval_min},{interval_max}]"
```

Prima di dividere il set di dati in training e validation set, escludiamo una parte dei dati da tenere per il test set e per bilanciare il numero di entry per classe.

Il test set ottenuto non sarà ottimale, avendo solo tre delle sei classi presenti e un numero non particolarmente alto di entry

(~20_000).
Tuttavia, come vedremo, sarà composto da un buon bilanciamento di classi facili e difficili da riconoscere.

In [34]:

```
balanced_df = df.copy(deep=True)

to_drop = some_by_genre(8_000, "rock").index
to_drop = to_drop.append(some_by_genre(11_000, "pop").index)
to_drop = to_drop.append(some_by_genre(2000, "hip hop").index)
to_drop = to_drop.append(some_by_genre(0, "r&b").index)
to_drop = to_drop.append(some_by_genre(0, "rap").index)
to_drop = to_drop.append(some_by_genre(0, "country").index)

balanced_df.drop(to_drop , inplace=True)
del to_drop
```

In [35]:

```
X_b_t, X_b_v, Y_b_t, Y_b_v = train_test_split(balanced_df['lyrics'], balanced_df['genre'], random_s
tate=42, test_size=1/3)
```

In [36]:

```
grid = {
    "model__C" : [.1, .5, 1, 2],
    "tokenizer__tokenizer" : [tokenize, tokenize_stem],
    "tokenizer__min_df" : [2, 5, 8, 10],
    "model__penalty": ["l1", "l2"],
    "model__class_weight": ["balanced", None]
}
```

In [70]:

```
logreg = Pipeline([
    ("tokenizer", TfidfVectorizer(ngram_range=(1,2))),
    ("model", LogisticRegression(solver = "saga", random_state = 12))
])

gs = GridSearchCV(logreg, param_grid=grid)
gs.fit(X_b_t, Y_b_t)
gs.score(X_b_v, Y_b_v)
```

Out[70]:

```
0.5225659948907181
```

Il modello migliore è quello suggerito da i parametri seguenti, ma il quarto modello della lista ha una precisione simile al primo ma tempi di training e score molto ridotti, dati dall'uso del tokenizer senza stemming.

Avrebbe anche anche l'effetto aggiuntivo di poter visualizzare le feature in modo chiaro, senza che siano state modificate dal processo di stemming.

Verifichiamo quindi che i due modelli siano simili confrontandoli

In [80]:

```
gs.best_params_
```

Out[80]:

```
{'model__C': 1,
 'model__class_weight': 'balanced',
 'model__penalty': 'l2',
 'tokenizer__min_df': 2,
 'tokenizer__tokenizer': <function __main__.tokenize_stem(words)>}
```

In [81]:

```
#print(Y_b_t.value_counts(normalize=True))
```

```
#Y_b_t.value_counts(normalize=True).plot.pie(autopct="%1.1f%%");
#Y_b_t.value_counts()
pd.DataFrame(gs.cv_results_).sort_values("mean_test_score", ascending=False)
```

Out[81]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_model__C	param_model__class_weight	param_model__pena
73	51.513650	1.233109	12.451421	0.808201	1	balanced	
105	51.240123	1.805299	11.648569	1.013565	2	balanced	
75	48.533767	2.080035	12.485893	1.242531	1	balanced	
72	22.474643	0.909664	4.527762	0.593512	1	balanced	
107	49.108341	1.229229	11.930579	0.611252	2	balanced	
77	47.472282	1.254427	11.495634	0.868557	1	balanced	
104	22.937909	0.770981	4.325980	0.172224	2	balanced	

In [83]:

```
logreg_best = Pipeline([
    ("tokenizer", TfidfVectorizer(ngram_range=(1,2), min_df=2, tokenizer=tokenize_stem)),
    ("model", LogisticRegression(solver = "saga", random_state = 12, C=1, class_weight="balanced",
penalty="l2"))
])

logreg_best.fit(X_b_t, Y_b_t)
lr_b_score = logreg_best.score(X_b_v, Y_b_v)
lr_b_score
```

Out[83]:

```
0.5189372526851328
```

In [39]:

```
logreg = Pipeline([
    ("tokenizer", TfidfVectorizer(ngram_range=(1,2), min_df=2, tokenizer=tokenize)),
    ("model", LogisticRegression(solver = "saga", random_state = 12, C=1, class_weight="balanced",
penalty="l2"))
])

logreg.fit(X_b_t, Y_b_t)
lr_score = logreg.score(X_b_v, Y_b_v)
lr_score
```

Out[39]:

```
0.5172413793103449
```

In [85]:

```
print(compare_models(lr_b_score, lr_score, len(Y_b_t), len(Y_b_t)))
```

```
La differenza fra i due modelli (confidenza 99%) è statisticamente non rilevante.
 Intervallo: [-0.01997746252742305,0.023369209276998926]
```

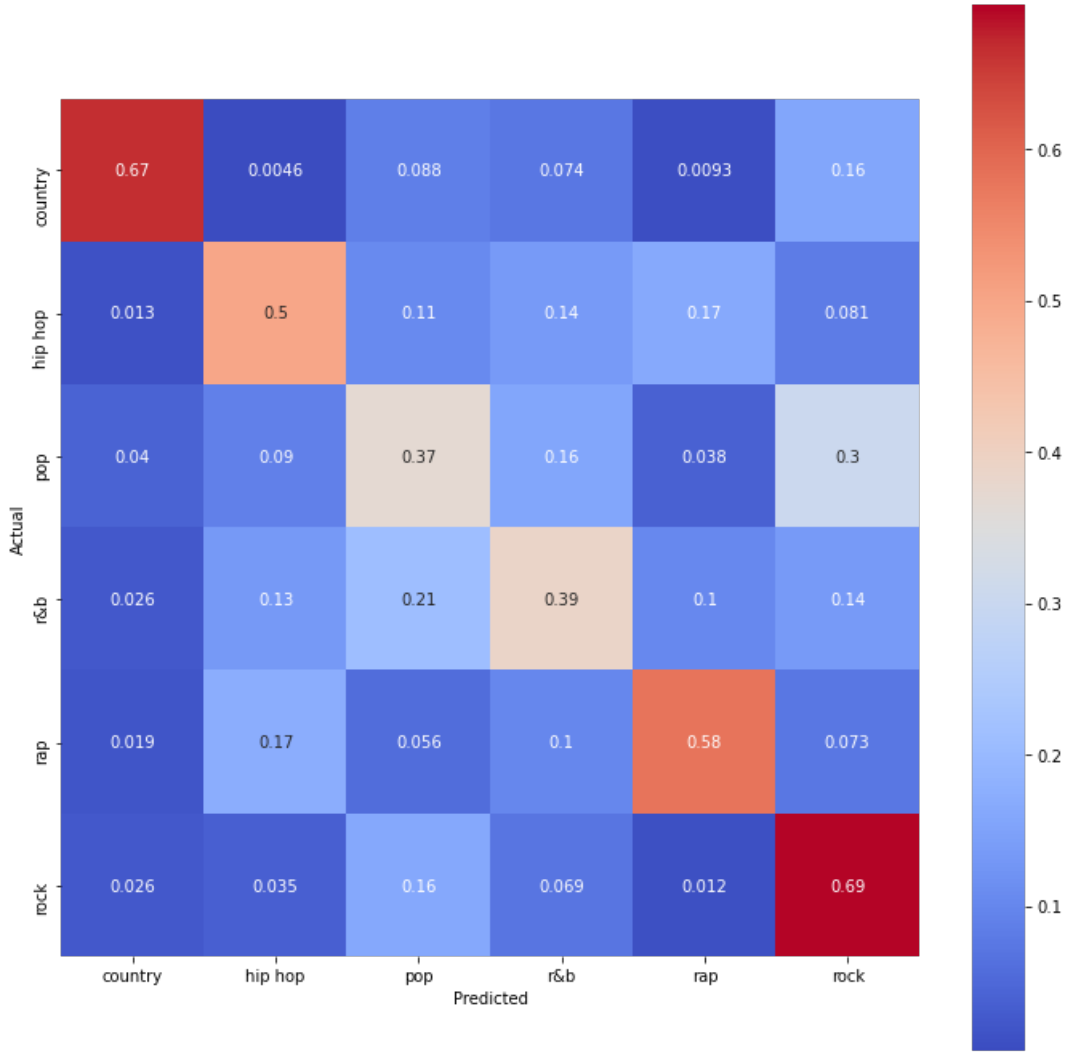
Vediamo che la maggior parte dei generi è abbastanza riconoscibile dal classificatore.

Analizzando la matrice di confusione si nota che:

- I generi catalogati con meno precisione sono il pop e il r&b
- Il pop è il genere che crea più confusione complessivmente, venendo spesso catalogato come rock e scelto come risultato quando invece le classi erano hip hop, r&b e rock.
  - Questo è comprensibile, essendo il pop un genere piuttosto vario.
- Il rythm and blue è abbastanza spesso scambiato per il hip hop: anche questo è comprensibile, essendo due generi con origine fortemente legata alla cultura afroamericana è probabile che abbiano parti di vocabolario condiviso
- Forse sorprendentemente, il rap viene confuso con l'hip hop solo nel 17% dei casi. E' un risultato accettabile, considerando che l'uno è un sottogenere dell'altro
- Come anticipato, il country ha una fortissima identità nei suoi testi, rispetto agli altri generi, anche questo è comprensibile

In [42]:

```
preds = logreg.predict(X_b_v)
draw_heat_map(preds, Y_b_v, logreg.classes_)
```



Osserviamo ora le features (le parole e i bi-gram) del modello con i relativi pesi (ordinati per importanza rispetto al country)

Si nota come, prevedibilmente, il country sia riconoscibile da termini campagnoli, l'hip hop e il rap da termini più forti e uso di slang e il pop e il rock da termini più popolari

In [43]:

```
pd.set_option("max_rows", None)
df_w_words = get_weighted_words(logreg.named_steps["model"],
                                logreg.named_steps["tokenizer"]).rename(columns=dict(enumerate(logr
g.classes_)))

df_w_words[-500:-1]
```

Out[43]:

	country	hip hop	pop	r&b	rap	rock
wrist	-0.255860	-0.087892	-0.244793	-0.090842	0.862658	-0.183271
ah ah	-0.256527	0.205348	0.639356	-0.255390	-0.262020	-0.070766
bit little	-0.256744	0.055610	-0.075239	0.014021	0.359617	-0.097265
check	-0.256835	0.228064	-0.259777	-0.081310	0.934594	-0.564736
loose	-0.257089	0.433135	0.328749	0.080688	-0.190417	-0.395065
silence	-0.257245	-0.284166	0.581009	-0.234964	-0.067240	0.262606
ta go	-0.257453	0.197075	-0.039820	0.028499	0.070607	0.001092
bum	-0.257794	0.129628	-0.235329	0.756787	-0.090962	-0.302330
like baby	-0.257805	0.247001	0.223709	0.157395	-0.181189	-0.189111
dont play	-0.257888	-0.187474	-0.041727	0.090337	0.343501	0.053249
way make	-0.258081	0.013866	0.090846	0.415848	-0.204027	-0.058452
person	-0.258650	0.160862	-0.164701	0.004281	0.219675	0.038533

In [44]:

```
df_w_words[:500]
```

Out[44]:

	country	hip hop	pop	r&b	rap	rock
little	4.882432	-0.641807	-0.643737	-1.802353	-1.733928	-0.060607
country	3.594520	-0.767357	-0.885122	-1.079814	-0.155878	-0.706350
old	3.344313	-0.836958	-1.015315	-1.136945	-0.594968	0.239874
good	2.332110	-0.638660	-0.693124	0.236307	-0.340794	-0.895839
said	2.019923	0.178426	-1.093759	-0.615168	-0.711509	0.222087
gon na	1.781515	-0.770770	0.658785	-1.148076	-1.674267	1.152812
road	1.752117	-0.344700	-0.628164	-0.962205	0.007017	0.175936
going	1.729775	0.180258	-0.179270	-1.114424	-0.169957	-0.446382
sitting	1.660756	-0.358911	-0.718210	-0.232451	-0.055937	-0.295248
cowboy	1.655581	-0.434043	-0.114135	-0.506663	-0.320187	-0.280554
aint	1.624627	1.325727	-2.368370	0.222359	1.894667	-2.699011

Test del modello a regime

Testiamo quindi il modello appreso sul test set.

Come osservato in precedenza, il test set non è stato definito nel modo migliore, ma contiene comunque un insieme bilanciato di classi di varia difficoltà (~2000 rock, ~3000 pop, ~2000 hip hop).

In [45]:

```
test_set = df[~df.index.isin(X_b_t.index) &
              ~df.index.isin(X_b_v.index) &
              ~df["genre"].isin(["edm"])]
```

In [47]:

```
X_test = test_set["lyrics"]
Y_test = test_set["genre"]
logreg_score = logreg.score(X_test, Y_test)
print(logreg_score)
```

```
0.5024432970680435
```

In [55]:

```
#confidenza 95%
Z = 1.96
N = len(Y_test)
base = (2 * N * logreg_score + Z**2) / (2 * (N + Z**2))
var = Z * np.sqrt(Z**2 + 4*N*logreg_score - 4*N*logreg_score**2) / (2 * (N + Z**2))
score_min, score_max = base - var, base + var
print(f"Intervallo di punteggio con confidenza 95%: [{score_min}, {score_max}]")
```

```
Intervallo di punteggio con confidenza 95%: [0.49578963285735456, 0.5090960960028129]
```

La precisione del modello addestrato rientra correttamente in quella dell'intervallo rilevato con 95% di confidenza.

Confronto con altri modelli

Per valutare la bontà del modello addestrato, verrà confrontato con un paio di soluzioni naive: la prima è un classificatore di tipo "stratificato", la seconda è quella del classificatore casuale uniforme.

In [49]:

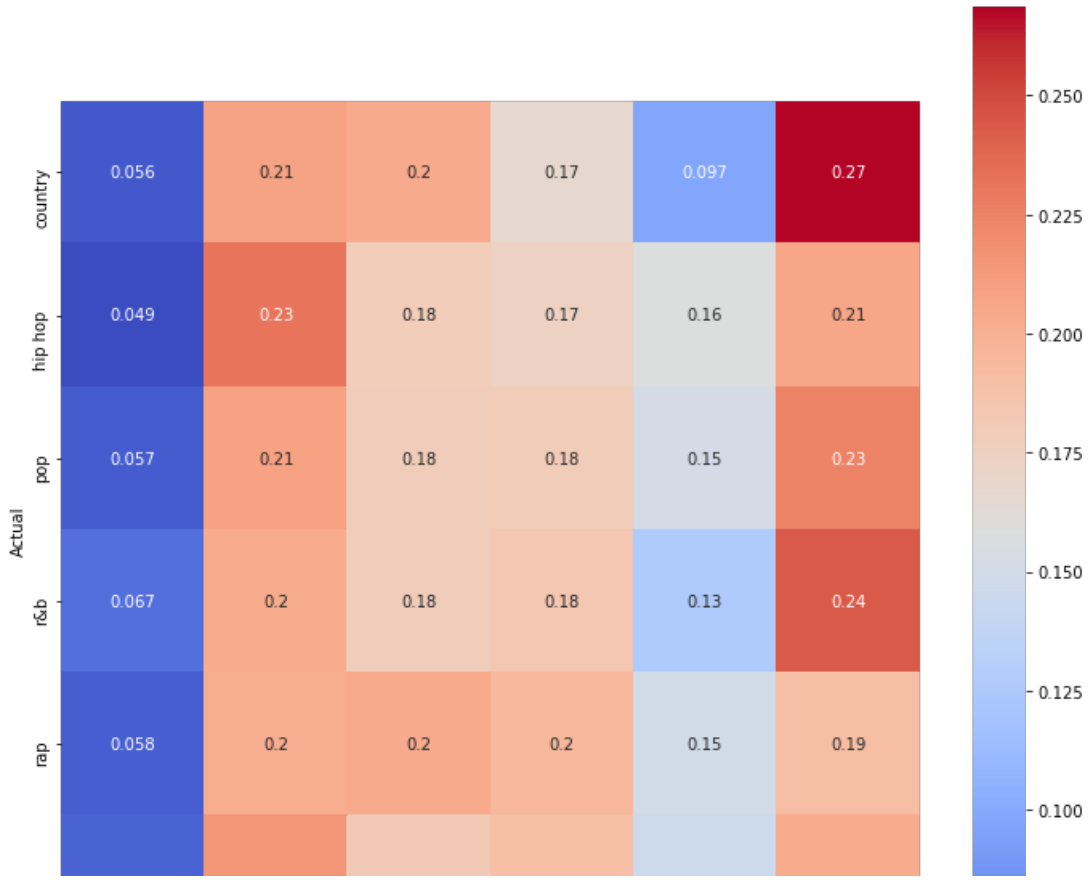
```
stratified = Pipeline([
    ("tokenizer", TfidfVectorizer(ngram_range=(1,2), min_df=2, tokenizer=tokenize)),
    ("model", DummyClassifier(strategy="stratified", random_state=12))
])
stratified.fit(X_b_t, Y_b_t)
stratified.score(X_b_v, Y_b_v)
```

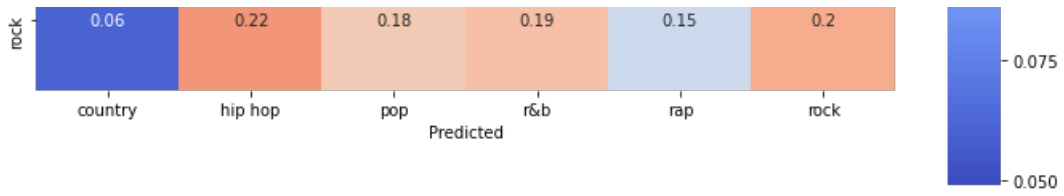
Out[49]:

```
0.18400226116449972
```

In [50]:

```
preds = stratified.predict(X_b_v)
draw_heat_map(preds, Y_b_v, stratified.classes_)
```



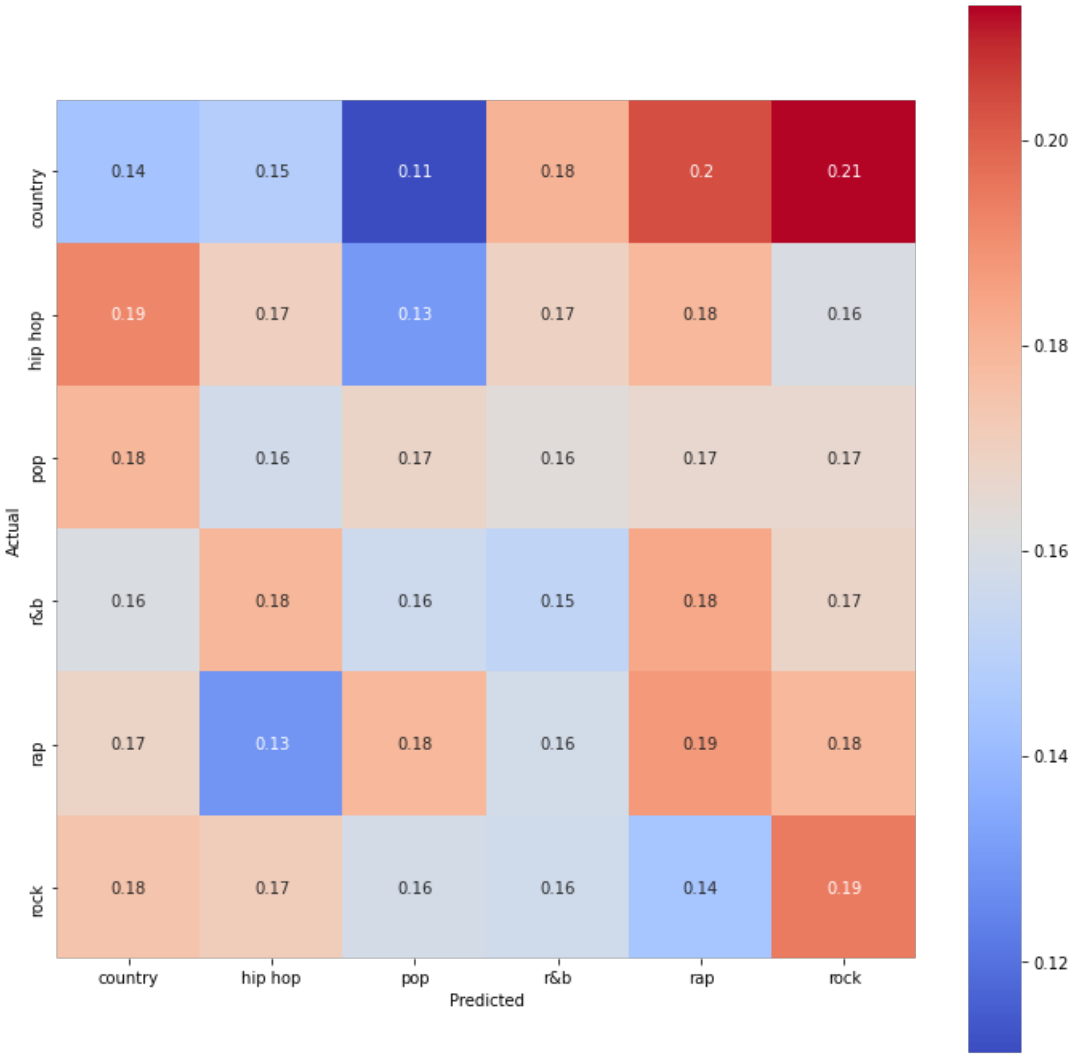


In [59]:

```
uniform = Pipeline([
    ("tokenizer", TfidfVectorizer(ngram_range=(1,2), min_df=2, tokenizer=tokenize)),
    ("model", DummyClassifier(strategy="uniform", random_state=12))
])
uniform.fit(X_b_t, Y_b_t)
uniform.score(X_b_v, Y_b_v)
```

In [53]:

```
preds = uniform.predict(X_b_v)
draw_heat_map(preds, Y_b_v, uniform.classes_)
```



In [62]:

```
stratified_score = stratified.score(X_b_t, Y_b_t)
uniform_score = uniform.score(X_b_t, Y_b_t)
logreg_score = logreg.score(X_b_t, Y_b_t)
```

In [80]:

```
print(compare_models(logreg_score, stratified_score, len(Y_b_t), len(Y_b_t)))
```

```
La differenza fra i due modelli (confidenza 99%) è statisticamente rilevante.
 Intervallo: [0.6784904513082416,0.7095819059557492]
```

In [81]:

```
print(compare_models(logreg_score, uniform_score, len(Y_b_t), len(Y_b_t)))
```

```
La differenza fra i due modelli (confidenza 99%) è statisticamente rilevante.
 Intervallo: [0.7035731305584603,0.733679554574383]
```

Conclusioni

I risultati ottenuti dal modello non sono certamente stati molto incoraggianti, ma pare che questo sia un problema normale per questo genere di studio, come evidenziato anche in questo studio https://medium.com/better-programming/predicting-a-songs-genre-using-natural-language-processing-7b354ed5bd80 e altri con i quali ho confrontato il mio elaborato.

"While some methods are better and more fitted for this program than others, the accuracy of the program never truly deviates from 60–70%. As we encountered several other machine learning projects to classify song lyrics, we found that most projects result in a similar range of accuracy."

Ho sperimentato anche io personalmente, durante lo svolgimento dell'elaborato, precisioni attorno al 60%, dovute o a dataset fortemente sbilanciati (per esempio, lasciando molte più istanze di pop e rock, questi due generi venivano categorizzati correttamente a discapito degli altri, ma i numeri sbilanciati davano l'illusione di un'accuratezza maggiore) oppure ad una scelta più netta dei generi da classificare: escludendo per esempio il pop nel mio caso, o accorpando i generi in modo differente, ho registrato anche picchi attorno al 66% di accuratezza (ho preferito studiare il comportamento del modello su più generi però).

Parte della ragione per il risultato è anche sicuramente da ricercare nei dataset, pieni di informazioni "sporche", in varie lingue e potenzialmente etichettate in modo sbagliato. Altro elemento particolarmente importante è stato la distribuzione delle classi: inserire troppe poche entry pop causava un crollo di precisione sulla categoria, mentre inserendone troppe si interferiva principalmente con rock, r&b e rap.

L'informazione chiave che si può ricavare è che i testi musicali sono solo in parte influenzati dal genere della canzone, soprattutto per quanto riguarda generi più popolari e meno definiti.