Project 2

Movie Genre Classification

Classify a movie genre based on its plot.

 https://www.kaggle.com/c/miia4200-20191-p2-moviegenreclassification/overview (https://www.kaggle.com/c/miia4200-20191-p2-moviegenreclassification/overview)

Data

Input:

- movie plot

Output: Probability of the movie belong to each genre

Evaluation

- 20% API
- 30% Create a solution using with a Machine Learning algorithm - Presentation (5 slides)
- 50% Performance in the Kaggle competition (Normalized acording to class performance in the private leaderboard)

Acknowledgements

We thank Professor Fabio Gonzalez, Ph.D. and his student John Arevalo for providing this dataset.

See https://arxiv.org/abs/1702.01992 (https://arxiv.org/abs/1702.01992)

Sample Submission

In [3]:
```
!pip install category_encoders
!pip install -U spacy
!python -m spacy download en_core_web_sm
!python -m spacy download en

import spacy
nlp = spacy.load("en")
```

```
Requirement already satisfied: category_encoders in /usr/local/envs/py3env/li
b/python3.5/site-packages (1.3.0)
Requirement already satisfied: statsmodels>=0.6.1 in /usr/local/envs/py3env/l
ib/python3.5/site-packages (from category_encoders) (0.8.0)
Requirement already satisfied: scipy>=0.17.0 in /usr/local/envs/py3env/lib/py
thon3.5/site-packages (from category_encoders) (1.0.0)
Requirement already satisfied: numpy>=1.11.1 in /usr/local/envs/py3env/lib/py
thon3.5/site-packages (from category_encoders) (1.16.3)
Requirement already satisfied: pandas>=0.20.1 in /usr/local/envs/py3env/lib/p
ython3.5/site-packages (from category_encoders) (0.22.0)
Requirement already satisfied: patsy>=0.4.1 in /usr/local/envs/py3env/lib/pyt
hon3.5/site-packages (from category_encoders) (0.5.0)
Requirement already satisfied: scikit-learn>=0.17.1 in /usr/local/envs/py3en
v/lib/python3.5/site-packages (from category_encoders) (0.19.1)
Requirement already satisfied: python-dateutil>=2 in /usr/local/envs/py3env/l
ib/python3.5/site-packages (from pandas>=0.20.1->category_encoders) (2.5.0)
Requirement already satisfied: pytz>=2011k in /usr/local/envs/py3env/lib/pyth
on3.5/site-packages (from pandas>=0.20.1->category_encoders) (2018.4)
Requirement already satisfied: six in /usr/local/envs/py3env/lib/python3.5/si
te-packages (from patsy>=0.4.1->category_encoders) (1.10.0)
Requirement already up-to-date: spacy in /usr/local/envs/py3env/lib/python3.
5/site-packages (2.1.3)
Requirement already satisfied, skipping upgrade: numpy>=1.15.0 in /usr/local/
envs/py3env/lib/python3.5/site-packages (from spacy) (1.16.3)
Requirement already satisfied, skipping upgrade: requests<3.0.0,>=2.13.0 in /
usr/local/envs/py3env/lib/python3.5/site-packages (from spacy) (2.18.4)
Requirement already satisfied, skipping upgrade: wasabi<1.1.0,>=0.2.0 in /us
r/local/envs/py3env/lib/python3.5/site-packages (from spacy) (0.2.1)
Requirement already satisfied, skipping upgrade: thinc<7.1.0,>=7.0.2 in /usr/
local/envs/py3env/lib/python3.5/site-packages (from spacy) (7.0.4)
Requirement already satisfied, skipping upgrade: plac<1.0.0,>=0.9.6 in /usr/l
ocal/envs/py3env/lib/python3.5/site-packages (from spacy) (0.9.6)
Requirement already satisfied, skipping upgrade: jsonschema<3.0.0,>=2.6.0 in
/usr/local/envs/py3env/lib/python3.5/site-packages (from spacy) (2.6.0)
Requirement already satisfied, skipping upgrade: cymem<2.1.0,>=2.0.2 in /usr/
local/envs/py3env/lib/python3.5/site-packages (from spacy) (2.0.2)
Requirement already satisfied, skipping upgrade: blis<0.3.0,>=0.2.2 in /usr/l
ocal/envs/py3env/lib/python3.5/site-packages (from spacy) (0.2.4)
Requirement already satisfied, skipping upgrade: preshed<2.1.0,>=2.0.1 in /us
r/local/envs/py3env/lib/python3.5/site-packages (from spacy) (2.0.1)
Requirement already satisfied, skipping upgrade: srsly<1.1.0,>=0.0.5 in /usr/
local/envs/py3env/lib/python3.5/site-packages (from spacy) (0.0.5)
Requirement already satisfied, skipping upgrade: murmurhash<1.1.0,>=0.28.0 in
/usr/local/envs/py3env/lib/python3.5/site-packages (from spacy) (1.0.2)
Requirement already satisfied, skipping upgrade: chardet<3.1.0,>=3.0.2 in /us
r/local/envs/py3env/lib/python3.5/site-packages (from requests<3.0.0,>=2.13.0
->spacy) (3.0.4)
Requirement already satisfied, skipping upgrade: idna<2.7,>=2.5 in /usr/loca
l/envs/py3env/lib/python3.5/site-packages (from requests<3.0.0,>=2.13.0->spac
y) (2.6)
Requirement already satisfied, skipping upgrade: urllib3<1.23,>=1.21.1 in /us
r/local/envs/py3env/lib/python3.5/site-packages (from requests<3.0.0,>=2.13.0
->spacy) (1.22)
Requirement already satisfied, skipping upgrade: certifi>=2017.4.17 in /usr/l
ocal/envs/py3env/lib/python3.5/site-packages (from requests<3.0.0,>=2.13.0->s
pacy) (2018.8.24)
Requirement already satisfied, skipping upgrade: tqdm<5.0.0,>=4.10.0 in /usr/
```

```
local/envs/py3env/lib/python3.5/site-packages (from thinc<7.1.0,>=7.0.2->spac
y) (4.31.1)
Requirement already satisfied: en_core_web_sm==2.1.0 from https://github.com/
explosion/spacy-models/releases/download/en_core_web_sm-2.1.0/en_core_web_sm-
2.1.0.tar.gz#egg=en_core_web_sm==2.1.0 in /usr/local/envs/py3env/lib/python3.
5/site-packages (2.1.0)
✓ Download and installation successful
You can now load the model via spacy.load('en_core_web_sm')
Requirement already satisfied: en_core_web_sm==2.1.0 from https://github.com/
explosion/spacy-models/releases/download/en_core_web_sm-2.1.0/en_core_web_sm-
2.1.0.tar.gz#egg=en_core_web_sm==2.1.0 in /usr/local/envs/py3env/lib/python3.
5/site-packages (2.1.0)
✓ Download and installation successful
You can now load the model via spacy.load('en_core_web_sm')
✓ Linking successful
/usr/local/envs/py3env/lib/python3.5/site-packages/en_core_web_sm -->
/usr/local/envs/py3env/lib/python3.5/site-packages/spacy/data/en
You can now load the model via spacy.load('en')
```

In [4]:
```
import pandas as pd
import os
import numpy as np
import category_encoders as ce
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.multiclass import OneVsRestClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import r2_score, roc_auc_score
from sklearn.model_selection import train_test_split, cross_val_score
import matplotlib.pyplot as plt
import xgboost as xgb
from nltk.stem import WordNetLemmatizer
from nltk.stem.snowball import SnowballStemmer
from sklearn.externals import joblib
import nltk
nltk.download('wordnet')
import re
```

```
/usr/local/envs/py3env/lib/python3.5/site-packages/sklearn/ensemble/weight_bo
osting.py:29: DeprecationWarning: numpy.core.umath_tests is an internal NumPy
module and should not be imported. It will be removed in a future NumPy relea
se.
  from numpy.core.umath_tests import inner1d
/usr/local/envs/py3env/lib/python3.5/site-packages/sklearn/cross_validation.p
y:41: DeprecationWarning: This module was deprecated in version 0.18 in favor
of the model_selection module into which all the refactored classes and funct
ions are moved. Also note that the interface of the new CV iterators are diff
erent from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)

[nltk_data] Downloading package wordnet to /content/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

In [5]:
```
dataTraining = pd.read_csv('https://github.com/albahnsen/PracticalMachineLearn
ingClass/raw/master/datasets/dataTraining.zip', encoding='UTF-8', index_col=0)
dataTesting = pd.read_csv('https://github.com/albahnsen/PracticalMachineLearni
ngClass/raw/master/datasets/dataTesting.zip', encoding='UTF-8', index_col=0)
```

In [6]:
```
dataTraining.head()
```

Out[6]:

	year	title	plot	genres	rating
3107	2003	Most	most is the story of a single father who takes...	['Short', 'Drama']	8.0
900	2008	How to Be a Serial Killer	a serial killer decides to teach the secrets o...	['Comedy', 'Crime', 'Horror']	5.6
6724	1941	A Woman's Face	in sweden , a female blackmailer with a disfi...	['Drama', 'Film-Noir', 'Thriller']	7.2
4704	1954	Executive Suite	in a friday afternoon in new york , the presi...	['Drama']	7.4
2582	1990	Narrow Margin	in los angeles , the editor of a publishing h...	['Action', 'Crime', 'Thriller']	6.6

In [7]:
```
dataTesting.head()
```

Out[7]:

	year	title	plot
1	1999	Message in a Bottle	who meets by fate , shall be sealed by fate ....
4	1978	Midnight Express	the true story of billy hayes , an american c...
5	1996	Primal Fear	martin vail left the chicago da ' s office to ...
6	1950	Crisis	husband and wife americans dr . eugene and mr...
7	1959	The Tingler	the coroner and scientist dr . warren chapin ...

Create y

In [8]:
```
dataTraining['genres'] = dataTraining['genres'].map(lambda x: eval(x))
le = MultiLabelBinarizer()
y_genres = le.fit_transform(dataTraining['genres'])
print(y_genres.shape)
y_genres
```

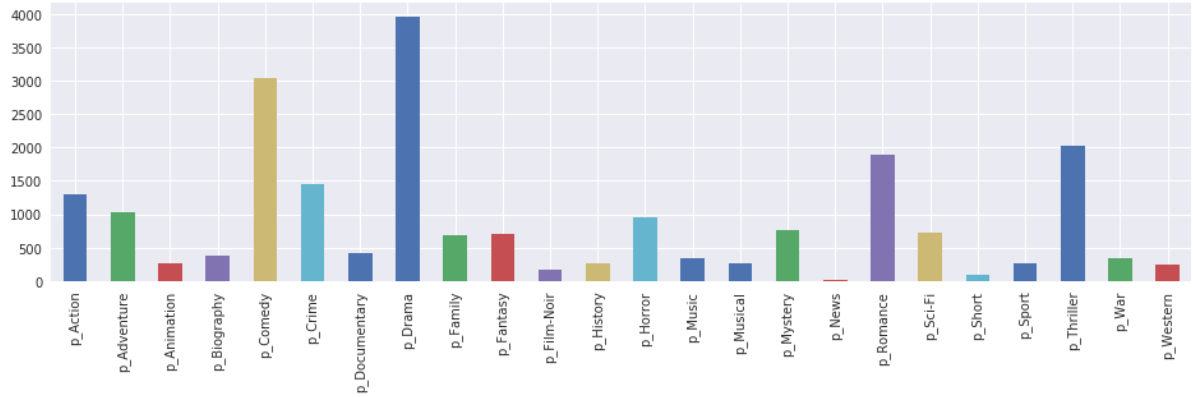
```
(7895, 24)
```

Out[8]:
```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 1, 0, 0],
       ...,
       [0, 1, 0, ..., 0, 0, 0],
       [0, 1, 1, ..., 0, 0, 0],
       [0, 1, 1, ..., 0, 0, 0]])
```

In [9]:
```
cols = ['p_Action', 'p_Adventure', 'p_Animation', 'p_Biography', 'p_Comedy',
'p_Crime', 'p_Documentary', 'p_Drama', 'p_Family',
       'p_Fantasy', 'p_Film-Noir', 'p_History', 'p_Horror', 'p_Music', 'p_Mus
ical', 'p_Mystery', 'p_News', 'p_Romance',
       'p_Sci-Fi', 'p_Short', 'p_Sport', 'p_Thriller', 'p_War', 'p_Western']

#train_DFY = pd.concat([train_DFY, dataTraining[['year']].reset_index()], axis
=1)
# ploter = train_DFY.groupby("year").sum().iloc[:,:24]
# ploter.head()

train_DFY = pd.DataFrame(y_genres, columns=cols)
pd.Series(train_DFY.sum().plot(kind = "bar",figsize=(16,4),rot = 0))
plt.xticks(rotation=90)
train_DFY.sum()
```

Out[9]:
```
p_Action         1303
p_Adventure      1024
p_Animation       260
p_Biography       373
p_Comedy         3046
p_Crime          1447
p_Documentary     419
p_Drama          3965
p_Family          682
p_Fantasy         707
p_Film-Noir       168
p_History         273
p_Horror          954
p_Music           341
p_Musical         271
p_Mystery         759
p_News              7
p_Romance        1892
p_Sci-Fi          723
p_Short            92
p_Sport           261
p_Thriller       2024
p_War             348
p_Western         237
dtype: int64
```

```
/usr/local/envs/py3env/lib/python3.5/site-packages/matplotlib/font_manager.py:1320: UserWarning: findfont: Font family ['sans-serif'] not found. Falling back to DejaVu Sans
  (prop.get_family(), self.defaultFamily[fontext]))
```



Preprocesamiento y vectorización de texto

```
# This function is call by the count vectorizer
def text_clean(text, remove_stop_words=True):
    wordnet_lemmatizer = WordNetLemmatizer()
    stemmer = SnowballStemmer('english')
    document = text

    # Remove all the special characters
    document = re.sub(r'\W', ' ', document)

    # remove all single characters
    document = re.sub(r'\s+[a-zA-Z]\s+', ' ', document)

    # Remove single characters from the start
    document = re.sub(r'\^[a-zA-Z]\s+', ' ', document)

    # Substituting multiple spaces with single space
    document = re.sub(r'\s+', ' ', document, flags=re.I)

    # Removing prefixed 'b'
```

Limpieza de texto

- Remover Caracteres Especiales
- Remover Caracteres únicos
- Remover Puntuación
- Remover Stopwords y otras palabras erradas
- Stimming
- Lammatizer
- Red neuronal pre-entrenada part of the speach

```
def create_XFeatures(plot_clean, title_clean, year, indexer, useCount=True, fit=True):

    #year BinaryEncoder
    dataTraining.year = dataTraining.year.astype(str)
    if fit: YearBinaryEnco.fit(year)
    YearBinary = YearBinaryEnco.transform(year)
    #print("YearBinary: " + str(YearBinary.shape))
    joblib.dump(YearBinaryEnco, 'YearBinaryEnco.pkl', compress=3)


    #TfidfVectorizer plot
    if fit: tfidf_plot.fit(plot_clean)
    plot_tfidf_dtm = tfidf_plot.transform(plot_clean)
    print("plot_tfidf_dtm: " + str(plot_tfidf_dtm.shape))
    plot_feat_tfidf = pd.DataFrame(plot_tfidf_dtm.toarray(), columns=tfidf_plot.get_feature
    #print(plot_feat_tfidf.head())
    joblib.dump(tfidf_plot, 'tfidf_plot.pkl', compress=3)
```

Vectorización texto

- Binary Encoder (Año)
- Count vectorizer (1 para title y 1 para plot)
- TFIDF vectorizer (1 para title y 1 para plot)
- N_gramas
- Min_df
- Max_df
- max_features

In [11]:
```
# This function transform the text in order get ready data, remove stop words,
stimming, Lemmatisation and n_grams
# This function is call by the count vectorizer
def text_clean(text, remove_stop_words=True):
    wordnet_lemmatizer = WordNetLemmatizer()
    stemmer = SnowballStemmer('english')
    document = text

    # Remove all the special characters
    document = re.sub(r'\W', ' ', document)

    # remove all single characters
    document = re.sub(r'\s+[a-zA-Z]\s+', ' ', document)

    # Remove single characters from the start
    document = re.sub(r'\^[a-zA-Z]\s+', ' ', document)

    # Substituting multiple spaces with single space
    document = re.sub(r'\s+', ' ', document, flags=re.I)

    # Removing prefixed 'b'
    document = re.sub(r'^b\s+', '', document)

    #Removing punctuation
    document = re.sub(r'[^\w\s]', '', document)

    #LowerCase
    document = document.lower()

    #Split document word a word
    #words_document = text.split()

    words_document = []
    doc = nlp(document)
    for token in doc: words_document.append(token.lemma_)



    #Remove clean_words
    words_document = [word for word in words_document if word not in waste_wor
ds]
    #Remove stop words
    if remove_stop_words:
        words_document = [word for word in words_document if word not in custo
m_stopwords]

    #stimming
    words_document = [stemmer.stem(word) for word in words_document]

    #Lemmatisation
    words_document = [wordnet_lemmatizer.lemmatize(word) for word in words_doc
ument]
    words_document = [wordnet_lemmatizer.lemmatize(word, pos='v') for word in
words_document]

    return ' '.join(words_document)
```

```
# The next lines are just for test the function
var = dataTraining.iloc[1,2]
#var = dataTraining.iloc[1,2] + " "+ dataTraining.iloc[1,1] + " "+str(dataTrai
ning.iloc[1,0])
print(var)
text_clean(var, remove_stop_words=True)
```

```
a serial killer decides to teach the secrets of his satisfying career to a vi
deo store clerk .
```

Out[11]:
```
'serial killer decid teach secret -pron- satisfi career video store clerk'
```

In [12]:
```
#Pre - process plot texts
plot_clean = []
for text in dataTraining["plot"]:
    plot_clean.append(text_clean(text, remove_stop_words=False))

#Pre - process title texts
title_clean = []
for text in dataTraining["title"]:
    title_clean.append(text_clean(text, remove_stop_words=False))
```

Build the features

Create count vectorizer function

In [13]:
```
YearBinaryEnco = ce.BinaryEncoder()
tfidf_plot = TfidfVectorizer()
tfidf_title = TfidfVectorizer()
vect_plot = CountVectorizer()
vect_title = CountVectorizer()

def create_XFeatures(plot_clean, title_clean, year, indexer, useCount=True, fi
t=True):

    #year BinaryEncoder
    dataTraining.year = dataTraining.year.astype(str)
    if fit: YearBinaryEnco.fit(year)
    YearBinary = YearBinaryEnco.transform(year)
    #print("YearBinary: " + str(YearBinary.shape))
    joblib.dump(YearBinaryEnco, 'YearBinaryEnco.pkl', compress=3)


    #TfidfVectorizer plot
    if fit: tfidf_plot.fit(plot_clean)
    plot_tfidf_dtm = tfidf_plot.transform(plot_clean)
    print("plot_tfidf_dtm: " + str(plot_tfidf_dtm.shape))
    plot_feat_tfidf = pd.DataFrame(plot_tfidf_dtm.toarray(), columns=tfidf_plo
t.get_feature_names(), index=indexer)
    #print(plot_feat_tfidf.head())
    joblib.dump(tfidf_plot, 'tfidf_plot.pkl', compress=3)

    #TfidfVectorizer title
    if fit: tfidf_title.fit(title_clean)
    title_tfidf_dtm = tfidf_title.transform(title_clean)
    print("title_tfidf_dtm: " + str(title_tfidf_dtm.shape))
    title_feat_tfidf = pd.DataFrame(title_tfidf_dtm.toarray(), columns=tfidf_t
itle.get_feature_names(), index=indexer)
    #print(title_feat_tfidf.head())
    joblib.dump(tfidf_title, 'tfidf_title.pkl', compress=3)

    if useCount:
        #CountVectorizer plot
        if fit: vect_plot.fit(plot_clean)
        plot_dtm = vect_plot.transform(plot_clean)
        print("plot_dtm: " + str(plot_dtm.shape))
        plot_features = pd.DataFrame(plot_dtm.toarray(), columns=vect_plot.get
_feature_names(), index=indexer)
        #print(plot_features.head())

        #CountVectorizer title
        if fit: vect_title.fit(title_clean)
        title_dtm = vect_title.transform(title_clean)
        print("title_dtm: " + str(title_dtm.shape))
        title_features = pd.DataFrame(title_dtm.toarray(), columns=vect_title.
get_feature_names(), index=indexer)
        #print(title_features.head())

        #concat all vectors
        return pd.concat([plot_feat_tfidf.add_suffix('_1'),
                          title_feat_tfidf.add_suffix('_2'),
                          plot_features.add_suffix('_3'),
```

```
                        title_features.add_suffix('_4'),
                        YearBinary], axis=1)

    return pd.concat([plot_feat_tfidf.add_suffix('_1'),
                      title_feat_tfidf.add_suffix('_2'),
                      YearBinary], axis=1)
```

Set the vectorizer functión parameters and run

In [14]:
```
#TD-IDF Vectorizer
tfidf_plot = TfidfVectorizer(ngram_range=(1, 3), max_features=12000,  min_df=7
, max_df=0.6)
tfidf_title = TfidfVectorizer(ngram_range=(1, 3), max_features=8000, min_df=2,
max_df=0.7)
#CountVectorizer
vect_plot = CountVectorizer(ngram_range=(1, 3), max_features=8000,  min_df=7,
max_df=0.5)
vect_title = CountVectorizer(ngram_range=(1, 3), max_features=8000,  min_df=2,
max_df=0.5)
#Year Binary encode
YearBinaryEnco = ce.BinaryEncoder()

#Fit the vectorizers and run with the clean data
X_features = create_XFeatures(plot_clean, title_clean, dataTraining[["year"]],
dataTraining.index, True, True)
print(X_features.shape)
#X_features.head()

# YearBinary: (7895, 8)
# plot_tfidf_dtm: (7895, 5297)
# title_tfidf_dtm: (7895, 4310)
# plot_dtm: (7895, 5297)
# title_dtm: (7895, 4310)
# (7895, 19222)
```

```
plot_tfidf_dtm: (7895, 12000)
title_tfidf_dtm: (7895, 4238)
plot_dtm: (7895, 8000)
title_dtm: (7895, 4238)
(7895, 28477)
```

Split train and test

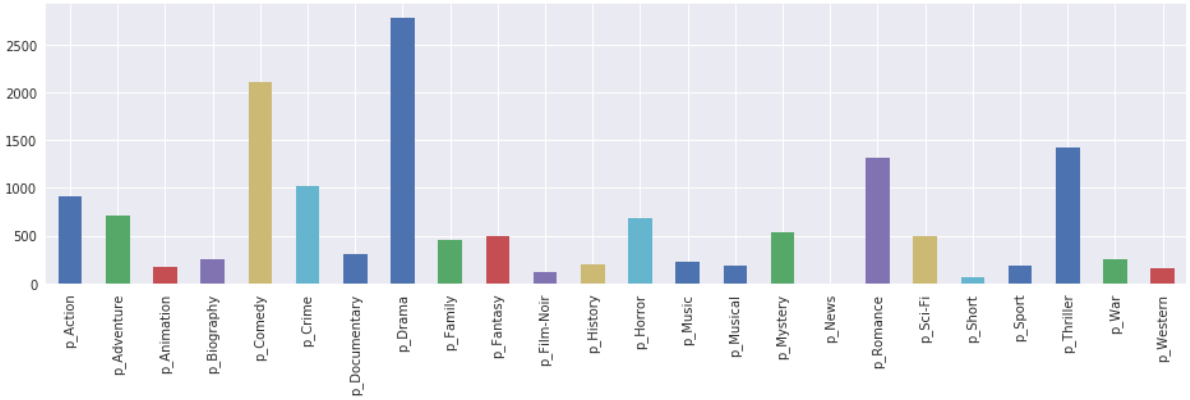
In [15]:
```
X_train, X_test, y_train_genres, y_test_genres = train_test_split(X_features,
y_genres, test_size=0.3, random_state=42)
X_train.shape
```

Out[15]:
```
(5526, 28477)
```

In [16]:
```
train_count_category = pd.DataFrame(y_train_genres, columns=cols, index=X_train.index)
pd.Series(train_count_category.sum().plot(kind = "bar",figsize=(16,4),rot = 0))
plt.xticks(rotation=90)
print(train_count_category.sum())
```

```
p_Action          916
p_Adventure       716
p_Animation       174
p_Biography       250
p_Comedy         2108
p_Crime          1020
p_Documentary     305
p_Drama          2787
p_Family          451
p_Fantasy         496
p_Film-Noir       114
p_History         203
p_Horror          688
p_Music           230
p_Musical         185
p_Mystery         537
p_News              4
p_Romance        1310
p_Sci-Fi          490
p_Short            65
p_Sport           180
p_Thriller       1426
p_War             252
p_Western         158
dtype: int64
```

```
/usr/local/envs/py3env/lib/python3.5/site-packages/matplotlib/font_manager.py:1320: UserWarning: findfont: Font family ['sans-serif'] not found. Falling back to DejaVu Sans
  (prop.get_family(), self.defaultFamily[fontext]))
```



Balancing clases

less than 800 rows oversampling to 800

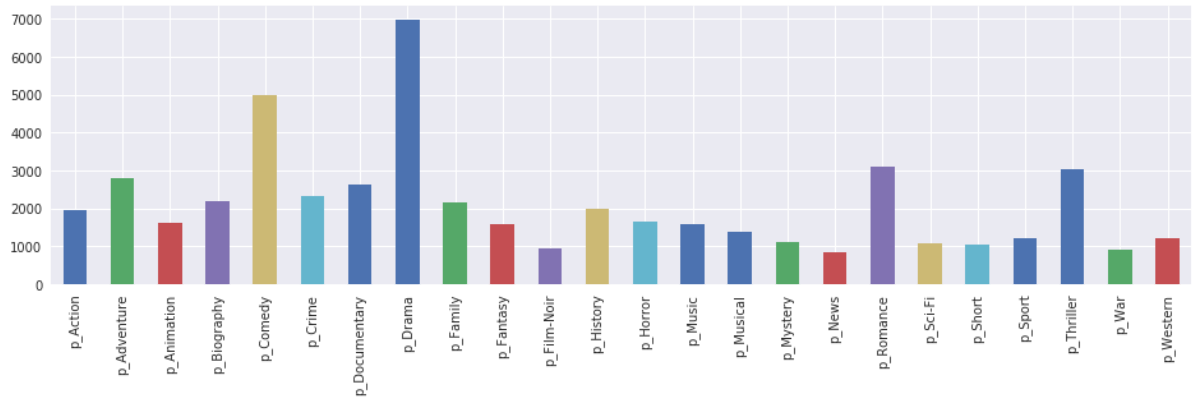
In [17]:
```
def overSampleData (X_features, y_categories ):
    overSampled_X = X_features
    train_count_category = pd.DataFrame(y_categories, columns=cols, index=X_tr
ain.index)
    overSampled_Y = train_count_category
    for i in range(1,24):
        if overSampled_Y.iloc[:,i].sum() < 800:
            #print("i=" + str(i) + " Cant: " + str(train_count_category.iloc
[:,i].sum()))
            OverSampleIds = np.random.choice(X_train[train_count_category.iloc
[:,i] == 1].index, 800)
            #print(dataTraining.loc[OverSampleIds].head())
            overSampled_X = pd.concat([overSampled_X, X_train.loc[OverSampleId
s]], axis=0)
            overSampled_Y = pd.concat([overSampled_Y, train_count_category.loc
[OverSampleIds]], axis=0)
            #print(overSampledDF.shape)
    return overSampled_X, overSampled_Y

overSampled_X, overSampled_Y = overSampleData(X_train, y_train_genres )
overSampled_X.shape
```

Out[17]: (15926, 28477)

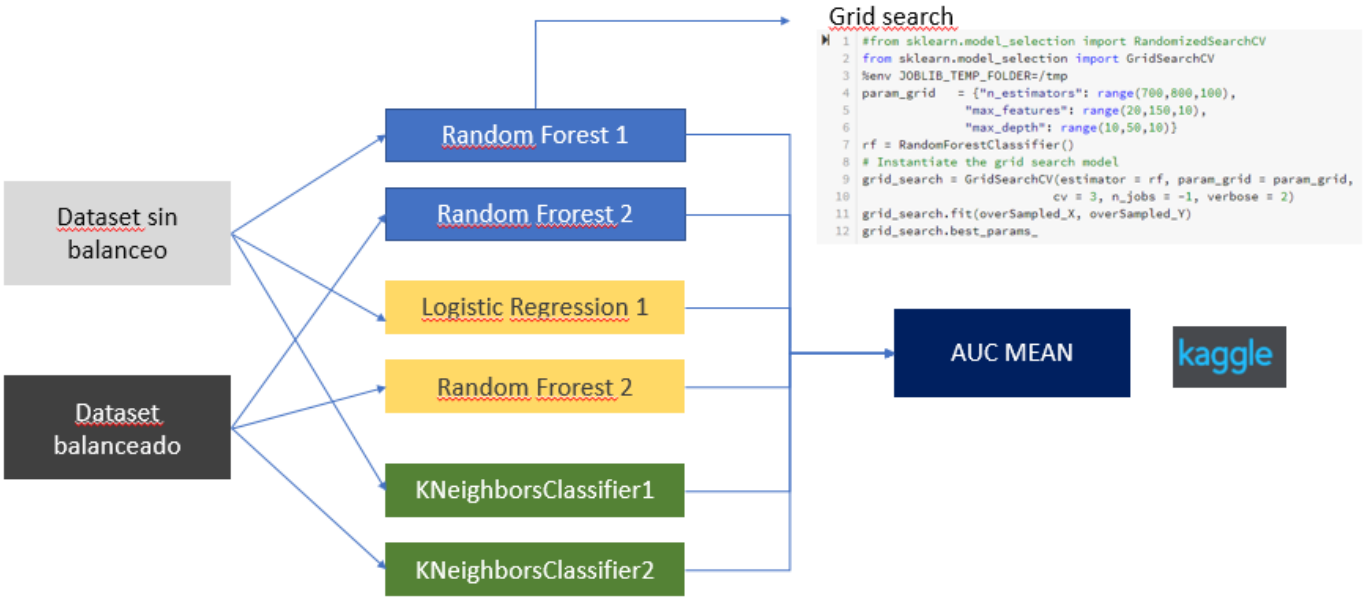
In [18]:
```
pd.Series(overSampled_Y.sum().plot(kind = "bar",figsize=(16,4),rot = 0))
plt.xticks(rotation=90)
print (overSampled_Y.sum())
```

```
p_Action          1960
p_Adventure       2783
p_Animation       1609
p_Biography       2198
p_Comedy          4985
p_Crime           2318
p_Documentary     2630
p_Drama           6988
p_Family          2147
p_Fantasy         1586
p_Film-Noir        937
p_History         1970
p_Horror          1648
p_Music           1569
p_Musical         1384
p_Mystery         1117
p_News             828
p_Romance         3112
p_Sci-Fi          1063
p_Short           1028
p_Sport           1196
p_Thriller        3041
p_War              888
p_Western         1199
dtype: int64
```

```
/usr/local/envs/py3env/lib/python3.5/site-packages/matplotlib/font_manager.p
y:1320: UserWarning: findfont: Font family ['sans-serif'] not found. Falling
back to DejaVu Sans
  (prop.get_family(), self.defaultFamily[fontext]))
```



Selección de modelos y parámetros



Random Forest

In [19]:
```
clf_over = OneVsRestClassifier(RandomForestClassifier(n_jobs=-1, n_estimators=
1000, max_features=140, max_depth=30, bootstrap=False, random_state=17))
clf_over.fit(overSampled_X, overSampled_Y)
```

Out[19]:
```
OneVsRestClassifier(estimator=RandomForestClassifier(bootstrap=False, class_w
eight=None, criterion='gini',
            max_depth=30, max_features=140, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=1000, n_jobs=-1,
            oob_score=False, random_state=17, verbose=0, warm_start=False),
          n_jobs=1)
```

In [20]:
```
y_pred_genres_rf_over = clf_over.predict_proba(X_test)
roc_auc_score(y_test_genres, y_pred_genres_rf_over, average='macro')
```

Out[20]:
```
0.8465730546203392
```

In [21]:
```
clf = OneVsRestClassifier(RandomForestClassifier(n_jobs=-1, n_estimators=1000,
max_features=140, max_depth=30, bootstrap=False, random_state=17))
clf.fit(X_train, y_train_genres )
```

Out[21]:
```
OneVsRestClassifier(estimator=RandomForestClassifier(bootstrap=False, class_w
eight=None, criterion='gini',
            max_depth=30, max_features=140, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=1000, n_jobs=-1,
            oob_score=False, random_state=17, verbose=0, warm_start=False),
          n_jobs=1)
```

In [22]:
```
y_pred_genres_rf = clf.predict_proba(X_test)
roc_auc_score(y_test_genres, y_pred_genres_rf, average='macro')
```

Out[22]:
```
0.8456644637049746
```

In [28]:
```
#Kaggle: 0.8542582153925989
#Last: 0.8576888447837199
```

Out[28]:
```
0.8485729516667776
```

Logistic Regression

In [23]:
```
lr_over =  OneVsRestClassifier(LogisticRegression())
lr_over.fit(overSampled_X, overSampled_Y)
```

Out[23]:
```
OneVsRestClassifier(estimator=LogisticRegression(C=1.0, class_weight=None, du
al=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False),
          n_jobs=1)
```

In [24]:
```
y_pred_genres_lr_over = lr_over.predict_proba(X_test)
```

In [25]:
```
roc_auc_score(y_test_genres, y_pred_genres_lr_over, average='macro')
```

Out[25]:
```
0.8377933501382552
```

In [26]:
```
lr =  OneVsRestClassifier(LogisticRegression())
lr.fit(X_train, y_train_genres)
```

Out[26]:
```
OneVsRestClassifier(estimator=LogisticRegression(C=1.0, class_weight=None, du
al=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False),
          n_jobs=1)
```

In [27]:
```
y_pred_genres_lr = lr.predict_proba(X_test)
roc_auc_score(y_test_genres, y_pred_genres_lr, average='macro')
```

Out[27]: 0.8426011508255021

Ensemble

In [28]:
```
y_pred_ensemb = y_pred_genres_rf_over
for i in range(0,len(y_test_genres)):
  for j in range(0,23):
    y_pred_ensemb[i][j] = (y_pred_genres_rf_over[i][j]+ y_pred_genres_rf[i][j]
+ y_pred_genres_lr_over[i][j] + y_pred_genres_lr[i][j])/4
```

In [29]:
```
roc_auc_score(y_test_genres, y_pred_ensemb, average='macro')
```

Out[29]: 0.8744587833549607

In [ ]:
```
kn_over = OneVsRestClassifier(KNeighborsClassifier())
kn_over.fit(overSampled_X, overSampled_Y)
```

In [ ]:
```
y_pred_genres_kn_over = kn_over.predict_proba(X_test)
roc_auc_score(y_test_genres, y_pred_genres_kn_over, average='macro')
```

In [ ]:
```
kn = OneVsRestClassifier(KNeighborsClassifier())
kn.fit(X_train, y_train_genres)
```

In [ ]:
```
y_pred_genres_kn = kn.predict_proba(X_test)
roc_auc_score(y_test_genres, y_pred_genres_kn, average='macro')
```

In [ ]:
```
y_pred_ensemb2 = [][]
for i in range(0,len(y_test_genres)):
  for j in range(0,23):
    y_pred_ensemb2[i][j] = (y_pred_genres_rf_over[i][j]+
                            y_pred_genres_rf[i][j] +
                            y_pred_genres_lr_over[i][j] +
                            y_pred_genres_lr[i][j] +
                            y_pred_genres_kn_over[i][j] +
                            y_pred_genres_kn [i][j])/6

roc_auc_score(y_test_genres, y_pred_ensemb2, average='macro')
```

Predict the testing dataset

In [30]:
```
## Train with full data

#clf_over.fit(overSampled_X, overSampled_Y)
clf.fit(X_features, y_genres)

#lr_over.fit(overSampled_X, overSampled_Y)
lr.fit(X_features, y_genres)

#kn_over.fit(overSampled_X, overSampled_Y)
#kn.fit(X_features, y_genres)
```

Out[30]:
```
OneVsRestClassifier(estimator=LogisticRegression(C=1.0, class_weight=None, du
al=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False),
          n_jobs=1)
```

In [31]:
```
#Pre - process title texts
title_test_clean = []
for text in dataTesting["title"]:
    title_test_clean.append(text_clean(text, remove_stop_words=False))

#Pre - process plot texts
plot_test_clean = []
for text in dataTesting["plot"]:
    plot_test_clean.append(text_clean(text))
```

In [32]:
```
#Fit with data train
X_features_test = create_XFeatures(plot_clean = plot_test_clean,
                                   title_clean = title_test_clean,
                                   year = dataTesting[["year"]],
                                   indexer = dataTesting.index,
                                   useCount = True,
                                   fit=False)
print(X_features_test.shape)
X_features_test.head()
```

```
plot_tfidf_dtm: (3383, 12000)
title_tfidf_dtm: (3383, 4238)
plot_dtm: (3383, 8000)
title_dtm: (3383, 4238)
(3383, 28477)
```

Out[32]:

	aaron_1	abandon_1	abandon pron_1	abbi_1	abduct_1	abduct by_1	abe_1	abigail_1	abil_1	abil to_1	...
1	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
4	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
5	0.357397	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
6	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
7	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...

5 rows × 28477 columns

In [34]:
```
#y_pred_test_genres = clf.predict_proba(X_features_test)

rf1 = clf_over.predict_proba(X_features_test)
rf2 = clf.predict_proba(X_features_test)

lr1 = lr_over.predict_proba(X_features_test)
lr2 = lr.predict_proba(X_features_test)
```

In [35]:
```
y_pred_test_genres = rf1
for i in range(0,len(y_pred_test_genres)):
    for j in range(0,23):
        y_pred_test_genres[i][j] = (rf1[i][j] +
                                    rf2[i][j] +
                                    lr1[i][j] +
                                    lr2[i][j] )/4
```

In [36]:
```
cols = ['p_Action', 'p_Adventure', 'p_Animation', 'p_Biography', 'p_Comedy',
'p_Crime', 'p_Documentary', 'p_Drama', 'p_Family',
       'p_Fantasy', 'p_Film-Noir', 'p_History', 'p_Horror', 'p_Music', 'p_Mus
ical', 'p_Mystery', 'p_News', 'p_Romance',
       'p_Sci-Fi', 'p_Short', 'p_Sport', 'p_Thriller', 'p_War', 'p_Western']

res = pd.DataFrame(y_pred_test_genres, index=dataTesting.index, columns=cols)
```

In [ ]:
```
res.head()
```

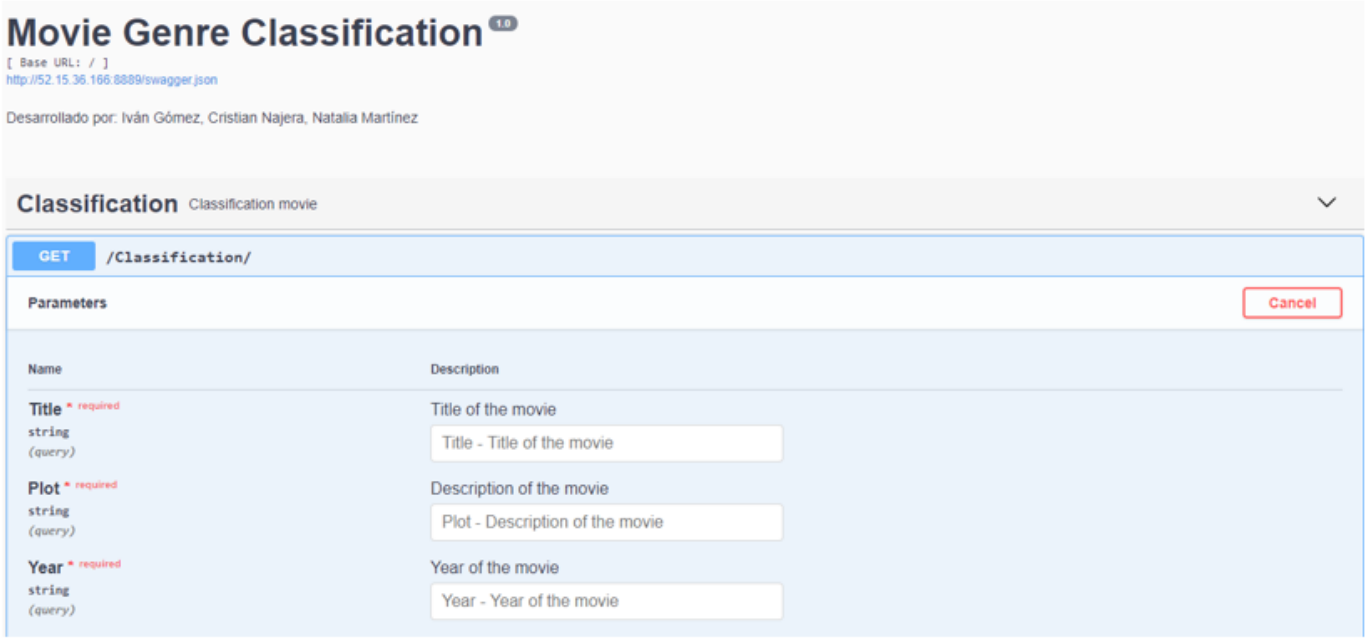
In [37]:
```
res.to_csv('pred_genres_text_RF.csv', index_label='ID')
```

http://52.15.36.166:8889/ (http://52.15.36.166:8889/)

http://52.15.36.166:8889/

Web Service

http://52.15.36.166:8889/



Stopwords

In [10]:
```
custom_stopwords =['a', 'about', 'above', 'across', 'after', 'afterwards', 'ag
ain',
       'against', 'ain', 'all', 'almost', 'alone', 'along', 'already',
       'also', 'although', 'always', 'am', 'among', 'amongst', 'amoungst',
       'amount', 'an', 'and', 'another', 'any', 'anyhow', 'anyone',
       'anything', 'anyway', 'anywhere', 'are', 'aren', "aren't",
       'around', 'as', 'at', 'back', 'be', 'became', 'because', 'become',
       'becomes', 'becoming', 'been', 'before', 'beforehand', 'behind',
       'being', 'below', 'beside', 'besides', 'between', 'beyond', 'bill',
       'both', 'bottom', 'but', 'by', 'call', 'can', 'cannot', 'cant',
       'co', 'con', 'could', 'couldn', "couldn't", 'couldnt', 'cry', 'd',
       'de', 'describe', 'detail', 'did', 'didn', "didn't", 'do', 'does',
       'doesn', "doesn't", 'doing', 'don', "don't", 'done', 'down', 'due',
       'during', 'each', 'eg', 'eight', 'either', 'eleven', 'else',
       'elsewhere', 'empty', 'enough', 'etc', 'even', 'ever', 'every',
       'everyone', 'everything', 'everywhere', 'except', 'few', 'fifteen',
       'fifty', 'fill', 'find', 'fire', 'first', 'five', 'for', 'former',
       'formerly', 'forty', 'found', 'four', 'from', 'front', 'full',
       'further', 'get', 'give', 'go', 'had', 'hadn', "hadn't", 'has',
       'hasn', "hasn't", 'hasnt', 'have', 'haven', "haven't", 'having',
       'he', 'hence', 'her', 'here', 'hereafter', 'hereby', 'herein',
       'hereupon', 'hers', 'herself', 'him', 'himself', 'his', 'how',
       'however', 'hundred', 'i', 'ie', 'if', 'in', 'inc', 'indeed',
       'interest', 'into', 'is', 'isn', "isn't", 'it', "it's", 'its',
       'itself', 'just', 'keep', 'last', 'latter', 'latterly', 'least',
       'less', 'll', 'ltd', 'm', 'ma', 'made', 'many', 'may', 'me',
       'meanwhile', 'might', 'mightn', "mightn't", 'mill', 'mine', 'more',
       'moreover', 'most', 'mostly', 'move', 'much', 'must', 'mustn',
      "mustn't", 'my', 'myself', 'name', 'namely', 'needn', "needn't",
       'neither', 'never', 'nevertheless', 'next', 'nine', 'no', 'nobody',
       'none', 'noone', 'nor', 'not', 'nothing', 'now', 'nowhere', 'o',
       'of', 'off', 'often', 'on', 'once', 'one', 'only', 'onto', 'or',
       'other', 'others', 'otherwise', 'our', 'ours', 'ourselves', 'out',
       'over', 'own', 'part', 'per', 'perhaps', 'please', 'put', 'rather',
       're', 's', 'same', 'see', 'seem', 'seemed', 'seeming', 'seems',
       'serious', 'several', 'shan', "shan't", 'she', "she's", 'should',
      "should've", 'shouldn', "shouldn't", 'show', 'side', 'since',
       'sincere', 'six', 'sixty', 'so', 'some', 'somehow', 'someone',
       'something', 'sometime', 'sometimes', 'somewhere', 'still', 'such',
       'system', 't', 'take', 'ten', 'than', 'that', "that'll", 'the',
       'their', 'theirs', 'them', 'themselves', 'then', 'thence', 'there',
       'thereafter', 'thereby', 'therefore', 'therein', 'thereupon',
       'these', 'they', 'thick', 'thin', 'third', 'this', 'those',
       'though', 'three', 'through', 'throughout', 'thru', 'thus', 'to',
       'together', 'too', 'top', 'toward', 'towards', 'twelve', 'twenty',
       'two', 'un', 'under', 'until', 'up', 'upon', 'us', 've', 'very',
       'via', 'was', 'wasn', "wasn't", 'we', 'well', 'were', 'weren',
      "weren't", 'what', 'whatever', 'when', 'whence', 'whenever',
       'where', 'whereafter', 'whereas', 'whereby', 'wherein',
       'whereupon', 'wherever', 'whether', 'which', 'while', 'whither',
       'who', 'whoever', 'whole', 'whom', 'whose', 'why', 'will', 'with',
       'within', 'without', 'won', "won't", 'would', 'wouldn', "wouldn't",
       'y', 'yet', 'you', "you'd", "you'll", "you're", "you've", 'your',
       'yours', 'yourself', 'yourselves']

waste_words =['!',
```

```
 '"',
 '$',
 '%',
 '&',
 "'",
 '(',
 ')',
 ',',
 '-',
 '.',
 '/',
 ':',
 ';',
 '=',
 '?',
 'a$$',
 'a&m',
 'aa',
 'aaa',
 'aam',
 '+',
 'aang']
```

xgboost

In [ ]:
```
# Train the model
xg_clf = OneVsRestClassifier(xgb.XGBClassifier(objective ='binary:logistic', m
ax_depth = 20, n_estimators = 600))
#xg_clf.fit(X_train, y_train)
xg_clf.fit(X_train, y_train_genres)
```

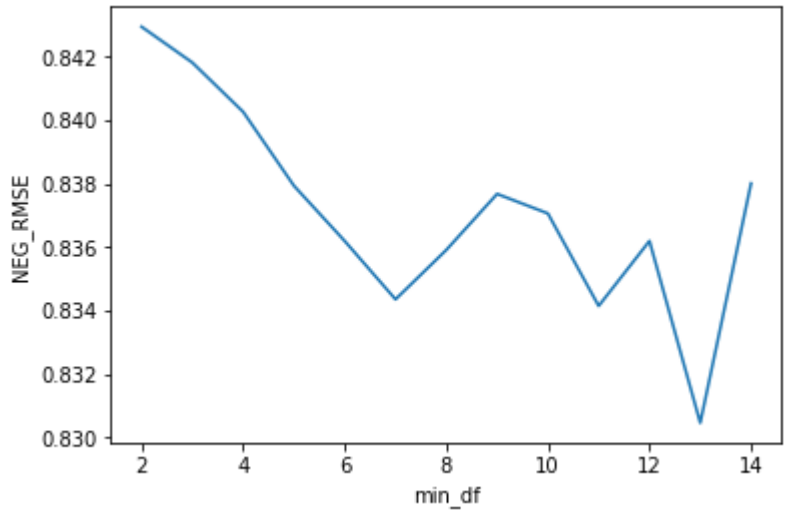
In [ ]:
```
y_pred_xgboost = xg_clf.predict(data_dmatrix_test)
y_pred_xgboost
```

In [ ]:
```
roc_auc_score(y_test_genres, y_pred_xgboost, average='macro')
```

In [18]:
```
# Cross validation to find the best parameters
range_for = range(2, 15, 1)
RMSE_scores_featu = []

for param in range_for:
    YearBinaryEnco = ce.BinaryEncoder()
    tfidf_plot = TfidfVectorizer(ngram_range=(1, (4+param)), max_features=7000
+(param*500),  min_df=12+param)
    tfidf_title = TfidfVectorizer(ngram_range=(1, (4+param)), max_features=700
0+(param*500), min_df=param)

    X_features = create_XFeatures(plot_clean, title_clean, dataTraining[["yea
r"]], dataTraining.index, False, True)
    print(X_features.shape)
    X_train, X_test, y_train_genres, y_test_genres = train_test_split(X_featur
es, y_genres, test_size=0.33, random_state=42)

    clf = OneVsRestClassifier(RandomForestClassifier(n_jobs=-1, n_estimators=6
00, max_features=14, max_depth=20,  random_state=42))
    RMSE_scores_featu.append(cross_val_score(clf, X_train, y_train_genres, cv=
3, scoring='roc_auc').mean())

best_param = range_for[RMSE_scores_featu.index(max(RMSE_scores_featu))]
print ("best_param: ", best_param)
plt.plot(range_for, RMSE_scores_featu)
plt.xlabel('min_df')
plt.ylabel('NEG_RMSE')
```

```
YearBinary: (7895, 8)
plot_tfidf_dtm: (7895, 4642)
title_tfidf_dtm: (7895, 4359)
(7895, 9009)
YearBinary: (7895, 8)
plot_tfidf_dtm: (7895, 4385)
title_tfidf_dtm: (7895, 2089)
(7895, 6482)
YearBinary: (7895, 8)
plot_tfidf_dtm: (7895, 4153)
title_tfidf_dtm: (7895, 1383)
(7895, 5544)
YearBinary: (7895, 8)
plot_tfidf_dtm: (7895, 3971)
title_tfidf_dtm: (7895, 1026)
(7895, 5005)
YearBinary: (7895, 8)
plot_tfidf_dtm: (7895, 3789)
title_tfidf_dtm: (7895, 781)
(7895, 4578)
YearBinary: (7895, 8)
plot_tfidf_dtm: (7895, 3629)
title_tfidf_dtm: (7895, 639)
(7895, 4276)
YearBinary: (7895, 8)
plot_tfidf_dtm: (7895, 3470)
title_tfidf_dtm: (7895, 517)
(7895, 3995)
YearBinary: (7895, 8)
plot_tfidf_dtm: (7895, 3325)
title_tfidf_dtm: (7895, 452)
(7895, 3785)
YearBinary: (7895, 8)
plot_tfidf_dtm: (7895, 3207)
title_tfidf_dtm: (7895, 389)
(7895, 3604)
YearBinary: (7895, 8)
plot_tfidf_dtm: (7895, 3077)
title_tfidf_dtm: (7895, 342)
(7895, 3427)
YearBinary: (7895, 8)
plot_tfidf_dtm: (7895, 2966)
title_tfidf_dtm: (7895, 292)
(7895, 3266)
YearBinary: (7895, 8)
plot_tfidf_dtm: (7895, 2857)
title_tfidf_dtm: (7895, 250)
(7895, 3115)
YearBinary: (7895, 8)
plot_tfidf_dtm: (7895, 2757)
title_tfidf_dtm: (7895, 220)
(7895, 2985)
best_param:  2
```

Out[18]: `Text(0, 0.5, 'NEG_RMSE')`



In [19]:
```
YearBinaryEnco = ce.BinaryEncoder()
tfidf_plot = TfidfVectorizer(ngram_range=(1, (4+best_param)), max_features=700
0+(best_param*500),  min_df=12+best_param)
tfidf_title = TfidfVectorizer(ngram_range=(1, (4+best_param)), max_features=70
00+(best_param*500), min_df=best_param)
vect_plot = CountVectorizer(ngram_range=(1, 4), max_features=7000,  min_df=12)
vect_title = CountVectorizer(ngram_range=(1, 4), max_features=7000,  min_df=2)

X_features = create_XFeatures(plot_clean, title_clean, dataTraining[["year"]],
dataTraining.index, True, True)
X_train, X_test, y_train_genres, y_test_genres = train_test_split(X_features,
y_genres, test_size=0.33, random_state=42)
```

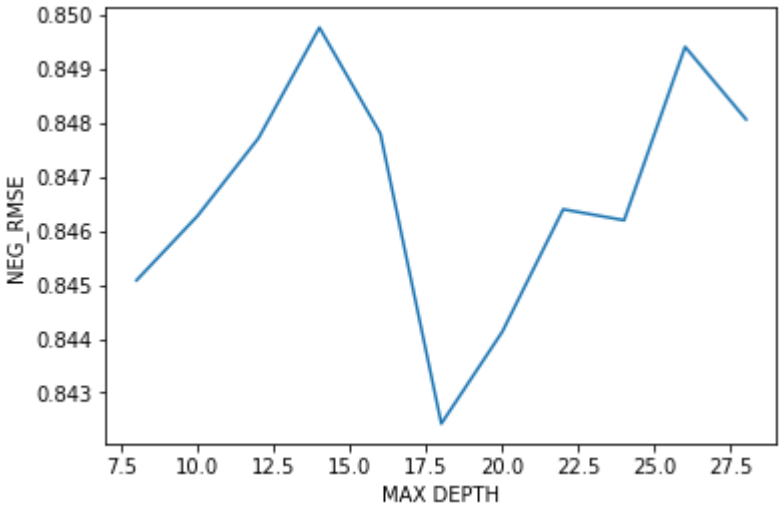
```
YearBinary: (7895, 8)
plot_tfidf_dtm: (7895, 4642)
title_tfidf_dtm: (7895, 4359)
plot_dtm: (7895, 5297)
title_dtm: (7895, 4310)
```

Train multi-class multi-label model Random Forest

In [20]:
```
# Cross validation to find the best depth parameter
max_depth_range = range(8, 30, 2)
RMSE_scores_featu = []

for depth in max_depth_range:
    clf = OneVsRestClassifier(RandomForestClassifier(n_jobs=-1, n_estimators=6
00, max_features=14, max_depth=depth,  random_state=17))
    RMSE_scores_featu.append(cross_val_score(clf, X_train, y_train_genres, cv=
3, scoring='roc_auc').mean())

best_maxdepth = max_depth_range[RMSE_scores_featu.index(max(RMSE_scores_featu
))]
print ("best_maxdepth: ", best_maxdepth)
plt.plot(max_depth_range, RMSE_scores_featu)
plt.xlabel('MAX DEPTH')
plt.ylabel('NEG_RMSE')
```

```
best_maxdepth:  14
```

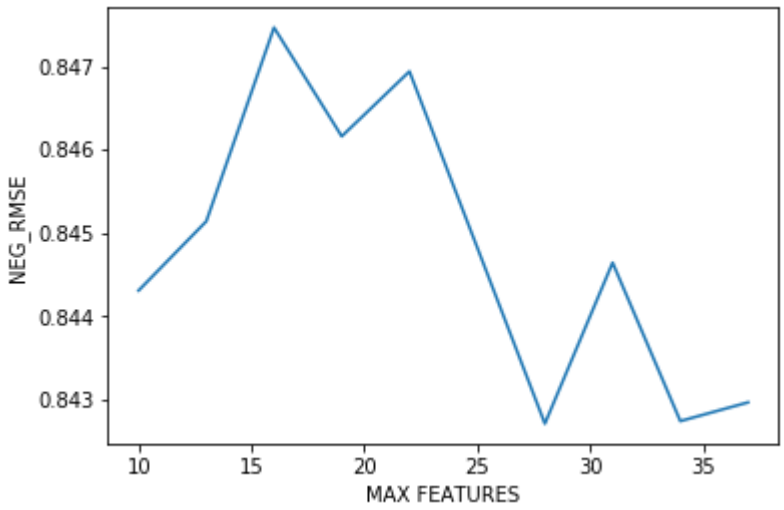
Out[20]:
```
Text(0, 0.5, 'NEG_RMSE')
```



In [27]:
```
# Cross validation to find the best max_features
max_features_range = range(10, 40, 3)
RMSE_scores_featu = []

for features in max_features_range:
    clf = OneVsRestClassifier(RandomForestClassifier(n_jobs=-1, n_estimators=6
00, max_features=features, max_depth=best_maxdepth,  random_state=17))
    RMSE_scores_featu.append(cross_val_score(clf, X_train, y_train_genres, cv=
3, scoring='roc_auc').mean())

best_max_features = max_features_range[RMSE_scores_featu.index(max(RMSE_scores
_featu))]
print ("best_max_features: ", best_max_features)
plt.plot(max_features_range, RMSE_scores_featu)
plt.xlabel('MAX FEATURES')
plt.ylabel('NEG_RMSE')
```

```
best_max_features:  16
```

Out[27]:
```
Text(0, 0.5, 'NEG_RMSE')
```



In [22]:
```
# Cross validation to find the best n_estimators
max_n_estimators = range(100, 900, 100)
RMSE_scores_featu = []

for estimators in max_n_estimators:
    clf = OneVsRestClassifier(RandomForestClassifier(n_jobs=-1, n_estimators=e
stimators, max_features=best_max_features, max_depth=best_maxdepth,  random_st
ate=17))
    RMSE_scores_featu.append(cross_val_score(clf, X_train, y_train_genres, cv=
3, scoring='roc_auc').mean())

best_n_estimators = max_depth_range[RMSE_scores_featu.index(max(RMSE_scores_fe
atu))]
print ("best_n_estimators: ", best_n_estimators)
plt.plot(max_n_estimators, RMSE_scores_featu)
plt.xlabel('N ESTIMATORS')
plt.ylabel('NEG_RMSE')
```

```
best_n_estimators:  22
```

Out[22]:
```
Text(0, 0.5, 'NEG_RMSE')
```