!!! N.B. : Toutes les informations ci-dessous sont extraites à partir du fichier fourni dans le livrable et datant du 09/05/2017.

Christophe Coudé, 2017

Parcours Data Scientist, projet n°3, rapport d'exploration

(basé sur les notebooks Projet3DataCleaning, Projet3ExplorationNum et Projet3ExplorationText)

Un site sur le cinéma souhaiterait lancer un moteur de recommandations de film pour sauver les soirées ciné de leurs futurs clients. Ce projet consiste donc à élaborer une API capable de retourner 5 recommandations de films similaires et intéressants pour le visiteur.

La base de donnée est un export (immuable du 30/08/2016) téléchargeable sur le site : https://www.kaggle.com/deepmatrix/imdb-5000-movie-dataset
et basé sur les données du site http://www.imdb.com

L'API doit être accessible via une requête obéissant au schéma :
GET {URL_DE_VOTRE_API}/recommend/{ID_FILM}

Le rendu doit être similaire à une réponse du type :

```
{
   "_results": [
      { "id": "645657", "name": "Eternal sunshine of the spotless mind" },
      { "id": "543556", "name": "500 Days of Summer" },
      { "id": "873453", "name": "Lost in Translation" }
   ]
}
```

Les variables présentes sont de différentes natures, on y trouve :

- des données textes (nom de film, d'acteurs, pays, langue etc.)
- une donnée url (lien vers la page IMDB du film)
- des données numériques (durée du film, popularité sur facebook du réalisateur et des 3 meilleurs actrices/acteur etc.)

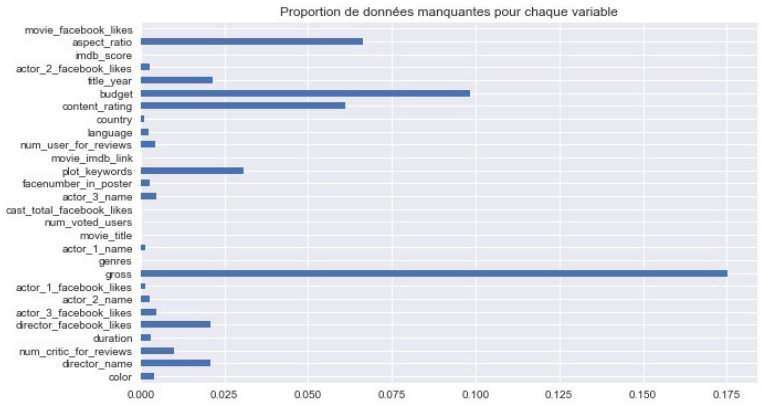
1- Gestion des données en double, manquantes et aberrantes, analyse univariée

In [2]:
```
print("Ce jeu de donnée comporte {0} enregistrements et {1} variables".format(dfOriginel.shape[0], dfOriginel.shape[1]))
print("Sur ces {0} enregistrements, on a {1} doublons".format(dfOriginel.shape[0], dfOriginel.shape[0] - dfOriginelSansDoublon.shape[0]))
```

```
Ce jeu de donnée comporte 5043 enregistrements et 28 variables
Sur ces 5043 enregistrements, on a 126 doublons
```

Ces doublons sont réellement des doublons dans le sens où ils marquent juste l'évolution du nombre de likes facebook d'un même film. On peut donc les supprimer et ne retenir que le dernier enregistrement.

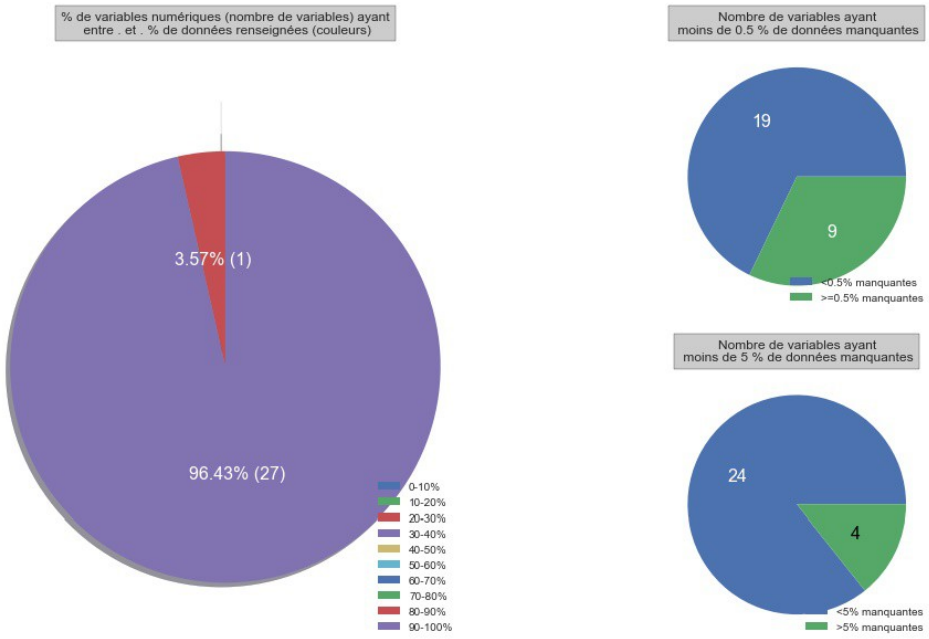
In [3]:
```
proportionDonneesManquantes(dfOriginelSansDoublon)
```



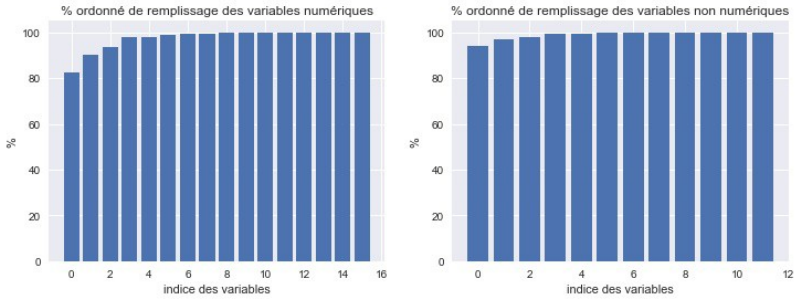
La proportion de données manquantes est, pour la plupart des variables, assez faible comme on peut le voir ci-dessus où le maximum se situe à environ 17.5 % de données manquantes pour la variable gross.
Ci-dessous, on peut voir que 27 des 28 variables (96.43%) sont remplies à plus de 90% et même 19 sur les 28 ont un taux de remplissage de plus de 99.5%.

In [4]:
```
graphCamembertRemplissage(dfOriginelSansDoublon)
```

```
<matplotlib.figure.Figure at 0xe4a32e8>
```



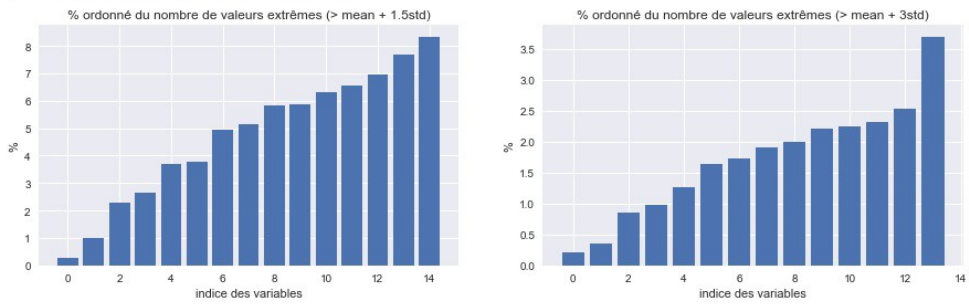
In [5]:
```
dfNumeriquesSD, dfNonNumeriquesSD = getDfNumerique(dfOriginelSansDoublon)
grapheDonneesManquantesVarNumNonNum(dfOriginelSansDoublon, dfNumeriquesSD, dfNonNumeriquesSD)
```



En ce qui concerne les valeurs extrêmes, elles ne sont pas nombreuses.

In [6]:
```
deuxGraphesExtremes(dfOriginelSansDoublon, dfNumeriquesSD)
```

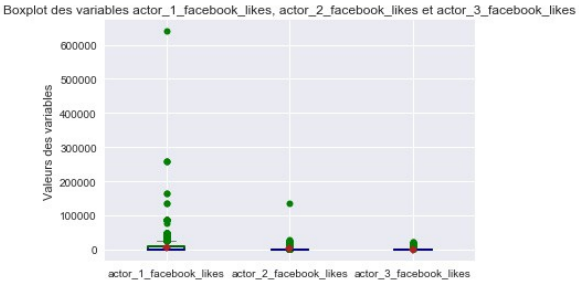
```
<matplotlib.figure.Figure at 0xce55fd0>
```



Et les boxplots n'ont révélé aucune valeur aberrante.

In [7]:
```
dfBoxPlot(dfNumeriquesSD[["actor_1_facebook_likes", "actor_2_facebook_likes", "actor_3_facebook_likes"]], "Boxplot des variables actor_1_facebook_likes, actor_2_facebook_likes et actor_3_facebook_likes")
```

```
<matplotlib.figure.Figure at 0xec94c50>
```

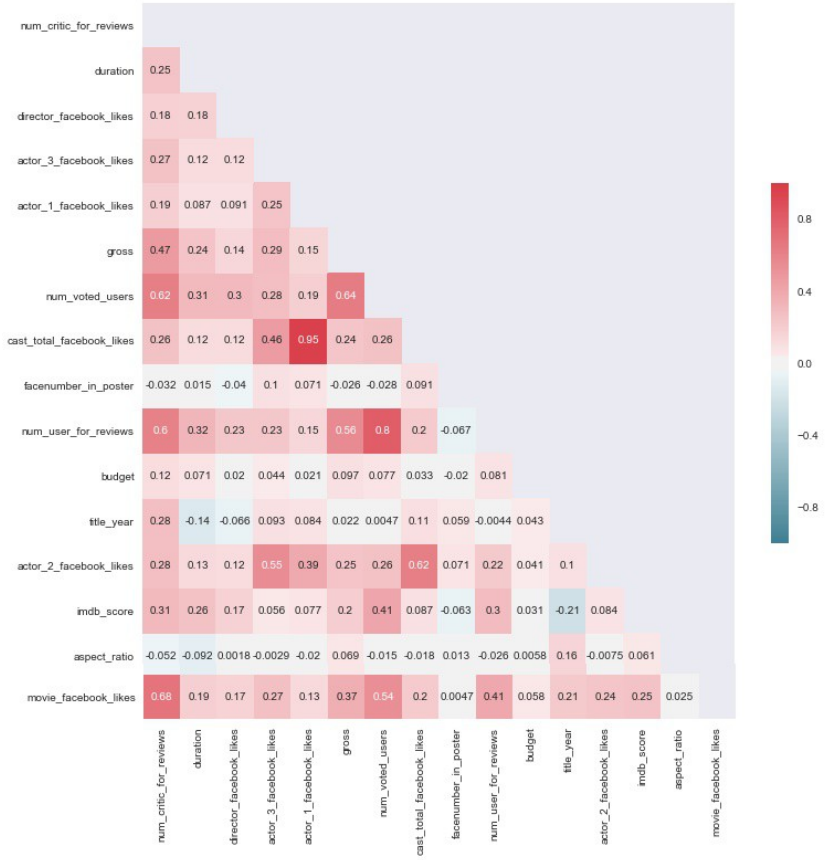


2- analyse multivariée

Heatmap de la matrice de corrélation des variables numériques initiales (sans one hot encoding par exemple)

In [8]:
```
heatmap(dfNumeriquesSD, "pearson")
```

```
<matplotlib.figure.Figure at 0x1055b710>
```



On voit d'après la heatmap des corrélations des variables numériques que mis à part la forte corrélation entre les variables cast_total_facebook_likes et actor_1_facebook_likes (qui est normale puisque l'une entre pour beaucoup dans la composition de l'autre), il n'y a pas de grosses corrélations. Il n'y a ainsi pas de corrélation à 1, on ne peut donc pas diminuer la complexité du dataset en supprimant une éventuelle variable.

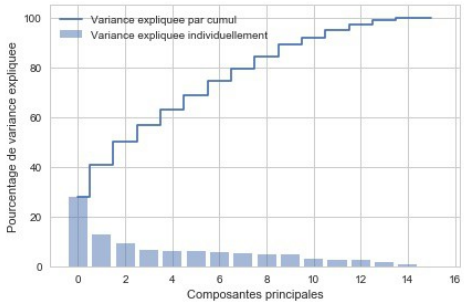
ACP sur les données sans doublons

In [9]:
```
dfNumeriquesSDSansDataManquant = dfNumeriquesSD.fillna(0)
dfNumeriquesSDSansDataManquantNormalise = prep.StandardScaler().fit_transform(dfNumeriquesSDSansDataManquant)
composantesPrincipale(dfNumeriquesSDSansDataManquantNormalise, dfNumeriquesSDSansDataManquant, 2, 0)
composantesPrincipale(dfNumeriquesSDSansDataManquantNormalise, dfNumeriquesSDSansDataManquant, 2, 1)
```

```
                      features     coeff  importances
6              num_voted_users  0.392367           11
0       num_critic_for_reviews  0.366312           10
9         num_user_for_reviews  0.356870           10
5                        gross  0.332060            9
7    cast_total_facebook_likes  0.283896            8
15        movie_facebook_likes  0.310645            8
3       actor_3_facebook_likes  0.250997            7
12      actor_2_facebook_likes  0.265849            7
4       actor_1_facebook_likes  0.225585            6
1                     duration  0.200782            5
                      features     coeff  importances
7    cast_total_facebook_likes  0.521996          146
4       actor_1_facebook_likes  0.487435          136
12      actor_2_facebook_likes  0.359737          100
3       actor_3_facebook_likes  0.266455           74
9         num_user_for_reviews -0.242934           68
6              num_voted_users -0.230379           64
13                  imdb_score -0.209004           58
0       num_critic_for_reviews -0.180471           50
8         facenumber_in_poster  0.170545           47
15        movie_facebook_likes -0.155325           43
```

In [10]:
```
eig_pairs = getPropre(dfNumeriquesSDSansDataManquantNormalise, True)
eig_vals_sorted = sorted(eig_pairs, key=lambda x:x[0], reverse=True)
print([eig_vals_sorted[u][0] for u in range(len(eig_vals_sorted))])
```

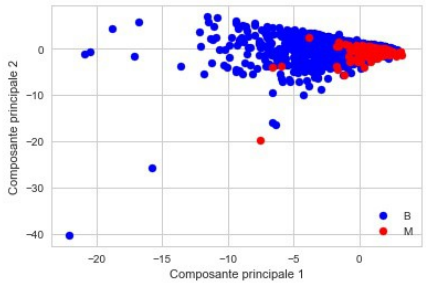
```
<matplotlib.figure.Figure at 0x1089e550>
```




```
[4.4550627310031423, 2.0709130010650814, 1.4764819765855999, 1.0620193087024508, 0.99793255924824742, 0.95835163796024936,
0.89502064198917775, 0.82348110617561898, 0.77363600135934252, 0.73588494028880136, 0.48134393685045929, 0.4294170222224461
5, 0.41088672939599635, 0.28062338590733926, 0.14733864871254151, 0.0016063725335031915]
```

In [11]:
```
predict_score = np.where(dfOriginelSansDoublon["imdb_score"]>5, "B", "M")
grapheACP2Composantes(eig_pairs, dfNumeriquesSDSansDataManquantNormalise, predict_score)
```

```
<matplotlib.figure.Figure at 0x109ee630>
```



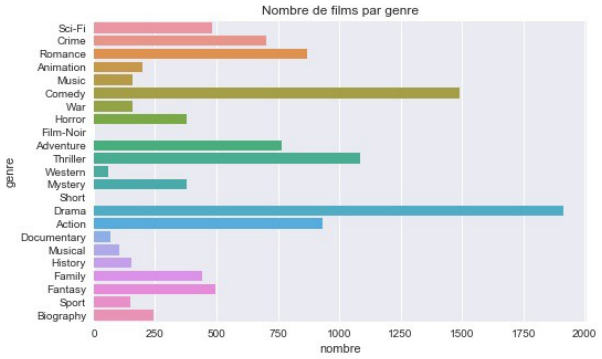
On peut noter deux choses vis-à-vis des résultats de l'ACP faite sur ces données :

- pour la première composante principale, la quantité de votes et de critiques est une notion importante pour le score. On peut peut-être l'interpréter dans le fait que plus un film sera connu et reconnu, plus le nombre de votant sera élevé, comme un effet d'entraînement, et qu'au contraire les films les moins reconnus susciteront le moins de votes et de critiques.
- pour la deuxième composante principale, ce sont les nombres de likes sur facebook qui ressortent. Toutefois, il n'y a pas de fortes corrélations entre ces variables et le score IMDB (cf. heatmap), ce qui se voit dans le fait que sur la composante principale 2, il ne semble pas possible de déterminer d'après le graphique si un film aura un bon score ou non en fonction de sa position sur cette composante principale. Cela semble indiquer que ces variables ne sont pas prédominentes, par exemple même les meilleurs acteurs ne peuvent pas empêcher un film d'être un navet, et réciproquement, un film avec des inconnus peut obtenir une très bonne note.

3- Quelques enseignements utilisant les données textes

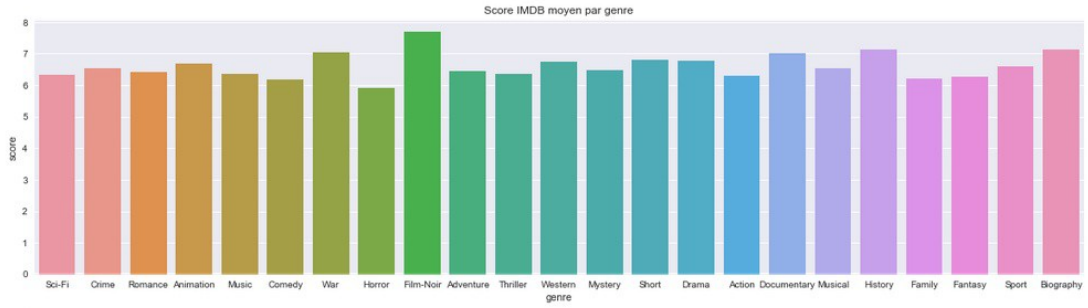
Nombre de films par genre

In [12]:
```
dfGenre, year, index, dfClean = getDfGenre(dfOriginelSansDoublon)
grapheNombreFilmParGenre(dfGenre)
```



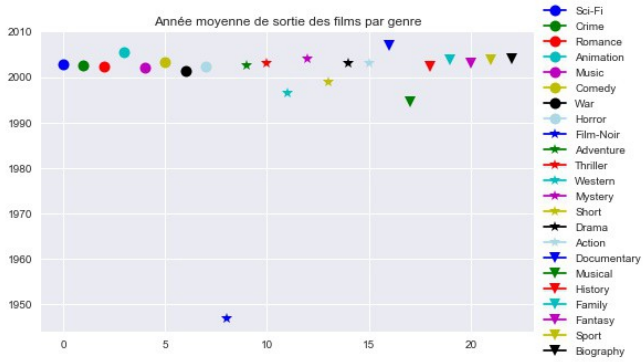
Score IMDB moyen par genre

In [13]:
```
grapheScoreFilmParGenre(dfGenre)
```



Année moyenne de sortie des films par genre

In [14]:
```
grapheAnneeMoyenneFilmParGenre(year, index)
```



Le site a été lancé au début des années 90, les éventuels films d'avant 1990 évalués ne sont pas majoritaires. "film-noir" ne semble pas être une catégorie où les films contemporains soient encore classés, ce qui semble normal puisque cette catégorie désigne habituellement des films français et américains des années 40-50 (ex. Le Faucon maltais, 1941, La soif du mal, 1958, source wikipedia)

Budget par genre et par année

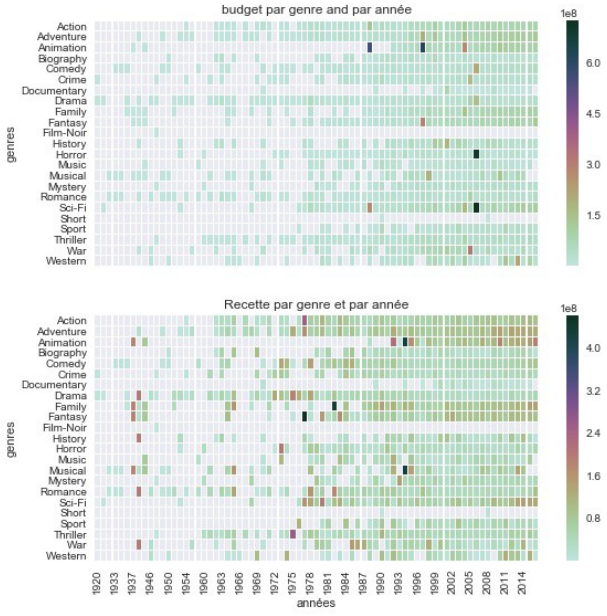
In [15]:
```
df_genre = pd.DataFrame(columns = ['genre', 'budget', 'gross', 'year'])

def genreRemap(row):
    global df_genre
    d = {}
    genres = np.array(row['genres'].split('|'))
    n = genres.size
    d['budget'] = [row['budget']]*n
    d['gross'] = [row['gross']]*n
    d['year'] = [row['title_year']]*n
    d['genre'] = []
    for genre in genres:
        d['genre'].append(genre)
    df_genre = df_genre.append(pd.DataFrame(d), ignore_index = True)
```

```
dfClean.apply(genreRemap, axis = 1)
df_genre['year'] = df_genre['year'].astype(np.int16)
df_genre = df_genre[['genre', 'budget', 'gross', 'year']]

genre_year = df_genre.groupby(['genre', 'year']).mean().reset_index()
df_gyBudget = genre_year.pivot_table(index = 'genre', columns = 'year', values = 'budget', aggfunc = np.mean)
df_gyGross = genre_year.pivot_table(index = 'genre', columns = 'year', values = 'gross', aggfunc = np.mean)
f, [axA, axB] = plt.subplots(figsize = (9, 9), nrows = 2)
cmap = sns.cubehelix_palette(start = 1.5, rot = 1.5, as_cmap = True)
sns.heatmap(df_gyBudget, xticklabels = 3, cmap = cmap, linewidths = 0.05, ax = axA)
sns.heatmap(df_gyGross, xticklabels = 3, cmap = cmap, linewidths = 0.05, ax = axB)
axA.set_title(u'budget par genre and par année')
axA.set_xlabel('')
axA.set_ylabel('genres')
axA.set_xticklabels([])
axB.set_title(u'Recette par genre et par année')
axB.set_xlabel(u'années')
axB.set_ylabel('genres')
```

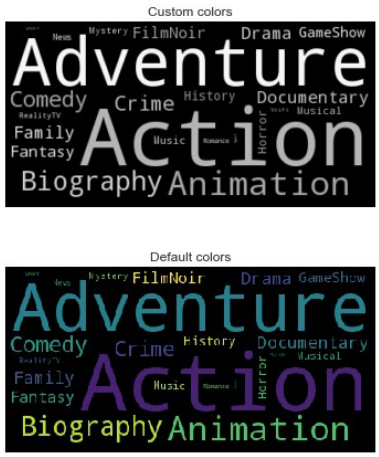
Out[15]: <matplotlib.text.Text at 0x6914e48>



On voit nettement l'augmentation des budgets et des recettes au fil des années et pour la plupart des genres.

Word Cloud des genres de films les plus représentés:

In [16]:
```
grapheWordCloud()
```



Les films d'action et d'aventure sont les plus représentés.

4- Méthode permettant de déterminer les 5 plus proches voisins d'un film

La méthode la plus immédiate pour déterminer les plus proches voisins d'un film et de considérer un film comme un vecteur dont les coordonnées sont les valeurs de ses variables, puis de déterminer à l'aide d'un algorithme les vecteurs qui lui sont les plus proches. On utilise alors la méthode NearestNeighbors avec différents algorithmes possibles :

Prenons par exemple le film "20 000 lieues sous les mers" du réalisateur Richard Fleischer (indice 3711 dans la liste des films initiale, c'est-à-dire avec doublons):

In [17]:
```
dfNumeriquesAD, _ = getDfNumerique(dfOriginel)
dfNumeriquesADSansDataManquant = dfNumeriquesAD.fillna(0)
dfNumeriquesADSansDataManquantNormalise = prep.StandardScaler().fit_transform(dfNumeriquesADSansDataManquant)
voisinsProches(6, "ball_tree", dfNumeriquesADSansDataManquantNormalise, 3711)
voisinsProches(6, "kd_tree", dfNumeriquesADSansDataManquantNormalise, 3711)
voisinsProches(6, "brute", dfNumeriquesADSansDataManquantNormalise, 3711)
voisinsProches(6, "auto", dfNumeriquesADSansDataManquantNormalise, 3711)
```

```
[3711 4894 3347 4162 3314 3852]
[  0.00000000e+00   8.67282773e-05   4.61274011e-01   6.09311677e-01
   6.09658628e-01   6.11973170e-01]
[3711 4894 3347 4162 3314 3852]
[  0.00000000e+00   8.67282773e-05   4.61274011e-01   6.09311677e-01
   6.09658628e-01   6.11973170e-01]
[3711 4894 3347 4162 3314 3852]
[  0.00000000e+00   8.67282785e-05   4.61274011e-01   6.09311677e-01
   6.09658628e-01   6.11973170e-01]
[3711 4894 3347 4162 3314 3852]
[  0.00000000e+00   8.67282773e-05   4.61274011e-01   6.09311677e-01
   6.09658628e-01   6.11973170e-01]
```

Les 4 algorithmes donnent les mêmes résultats. Par exemple, si on étudie les 5 plus proches voisins du film d'indice 3711 (ligne 3713 du .csv) "20,000 Leagues Under the Sea" :

On trouve bien le doublon comme premier voisin:

indice	movie_title	director_name	genres	actor_1_name	actor_1_facebook_likes	duration	gross	imdb_scor
3711	20,000 Leagues Under the Sea	Richard Fleischer	Adventure_Drama_Family_Fantasy_Sci-Fi	James Mason	617	127		7.
4894	20,000 Leagues Under the Sea	Richard Fleischer	Adventure_Drama_Family_Fantasy_Sci-Fi	James Mason	618	127		7.
3347	Limbo	John Sayles	Adventure_Drama_Thriller	Mary Elizabeth Mastrantonio	638	126	1997807	7.
4162	The Black Stallion	Carroll Ballard	Adventure_Family_Sport	Teri Garr	481	118		7.
3314	All or Nothing	Mike Leigh	Drama	Lesley Manville	149	121	112935	7.
3852	Salvador	Oliver Stone	Drama_History_Thriller_War	Jim Belushi	854	122		7.

In [18]:
```
print(dfOriginel.iloc[3711, 11:14])
print(dfOriginel.iloc[4894, 11:14])
```

```
movie_title                 20,000 Leagues Under the Sea
num_voted_users                                    22123
cast_total_facebook_likes                            799
Name: 3711, dtype: object
movie_title                 20,000 Leagues Under the Sea
num_voted_users                                    22124
cast_total_facebook_likes                            800
Name: 4894, dtype: object
```

Le film le plus proche du 3711 est le 4894, avec une distance de 0.0000867, il est vraiment très proche par rapport aux suivants (0.461, 0.609 etc.). C'est bien un doublon (seul évolue temporellement le nombre de votants ou de likes facebook).

On enlève désormais les doublons, c'est-à-dire qu'on ne garde que les doublons d'indice les plus élevés (les enregistrements les plus récents), ce qui modifie la liste des indices.
Par exemple, dans la liste avec doublons, le "20,000 Leagues Under the Sea" le plus récent est l'enregistrement d'indice 4894, l'enregistrement d'incide 3711 est supprimé dans la liste sans doublons.

In [19]:
```
getFilmNouvelIndice(4894)
```

Out[19]: 4769

Dans cette nouvelle liste sans doublons, le nouvel indice du film "20,000 Leagues Under the Sea" n'est plus le 4894 mais le 4769.

In [20]:
```
voisinsProches(6, "ball_tree", dfNumeriquesSDSansDataManquantNormalise, 4769)
voisinsProches(6, "kd_tree", dfNumeriquesSDSansDataManquantNormalise, 4769)
voisinsProches(6, "brute", dfNumeriquesSDSansDataManquantNormalise, 4769)
voisinsProches(6, "auto", dfNumeriquesSDSansDataManquantNormalise, 4769)
```

```
[4769 3230 3197 4038 3730 3956]
[ 0.          0.45935061  0.60654291  0.60803134  0.61308121  0.61787016]
[4769 3230 3197 4038 3730 3956]
[ 0.          0.45935061  0.60654291  0.60803134  0.61308121  0.61787016]
[4769 3230 3197 4038 3730 3956]
[ 0.          0.45935061  0.60654291  0.60803134  0.61308121  0.61787016]
[4769 3230 3197 4038 3730 3956]
[ 0.          0.45935061  0.60654291  0.60803134  0.61308121  0.61787016]
```

In [21]:
```
#[3711 4894 3347 4162 3314 3852]
#[4769      3230 3197 4038 3730 3956]
print(getFilmNouvelIndice(3711), getFilmNouvelIndice(3347), getFilmNouvelIndice(4162), getFilmNouvelIndice(3314), getFilmNouvelIndice(3852))
```

```
(4769, 3230, 4038, 3197, 3730)
```

On peut voir que sans les doublons, la liste des indices change, mais pourtant, ce sont bien les mêmes films, les indices ont changé du fait de la suppression des doublons. On retrouve bien les nouveaux indices en recherchant la correspondance des anciens indices dans la nouvelle liste. Il y a juste une petite inversion entre deux des positions (4038/3197) du fait de la proximité des distances entre ces films.

5- Essais pour diminuer la distance entre les films

Puisque le critère de hiérarchie entre les films est la distance entre ces films, utilisant en dernier recours la méthode NearestNeighbors, on peut chercher à voir s'il n'existe pas des moyens de diminuer cette distance.

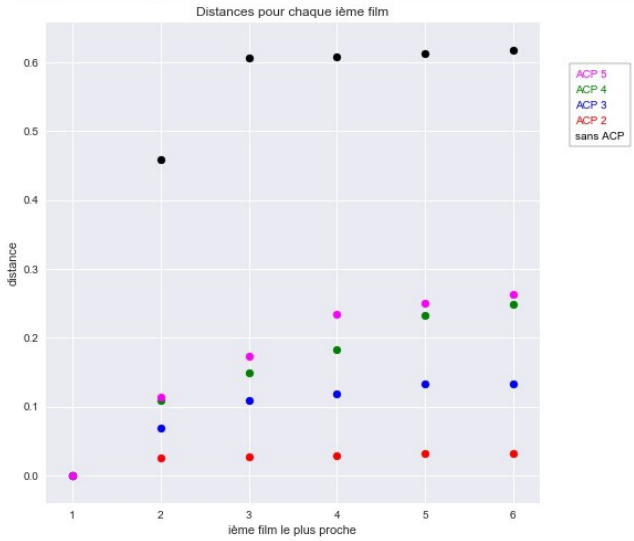
Utilisation des seules variables numériques

En utilisant une ACP, on diminue la distance entre les films:

In [22]:
```
tab_dist = []
tab_dist.append([0, 0.45935061, 0.60654291, 0.60803134, 0.61308121, 0.61787016])
for i in range(2,6):
    reduced_data = PCA(n_components=i).fit_transform(dfNumeriquesSDSansDataManquantNormalise)
    nbrs = NearestNeighbors(n_neighbors=6, algorithm="ball_tree").fit(reduced_data)
    distances, indices = nbrs.kneighbors(reduced_data)
    print(indices[4769])
    print(distances[4769])
    tab_dist.append(distances[4769])
```

```
[4769 3092 4374 3386 1983 1337]
[ 0.          0.02570104  0.02706365  0.02837312  0.03192473  0.03206773]
[4769 4417 3230 4829 3869 4235]
[ 0.          0.06838754  0.1084747   0.11894982  0.13255584  0.13286551]
[4769 3230 1990 4047 3143 2209]
[ 0.          0.10884718  0.149894    0.1830805   0.23327614  0.2492639 ]
[4769 3230 1990 3143 2209 3933]
[ 0.          0.1142873   0.17369312  0.23340386  0.24976392  0.26375328]
```

In [23]:
```
graphesDistACP(tab_dist)
```



Comme on peut le voir pour les 5 films les plus proches du 4769, en réduisant les données avec l'ACP, on obtient des distances moindres que sans l'ACP et les distances les plus basses ont lieu pour une ACP à 2 composantes principales.

Mais ce n'est qu'un exemple. On peut voir si la tendance est la même pour la moyenne des distances pour tous les films:

In [24]:
```
getMoyMoyACP2(dfNumeriquesSDSansDataManquantNormalise, "kd_tree")
```

```
Sans ACP, moyenne des distances pour les 5 films les plus proches :
[0.88489090094670964, 1.0182227658014549, 1.1018999640319718, 1.1629905076894018, 1.2088338276369639]
Sans ACP, moyenne des moyennes des distances des 5 films les plus proches : 1.07536759322

Avec ACP 2 composantes, moyenne des distances pour les 5 films les plus proches :
[0.056193640844021155, 0.086320402441337901, 0.10810276109964867, 0.12423997729814204, 0.13939587335409273]
Avec ACP 2 composantes, moyenne des moyennes des distances des 5 films les plus proches : 0.102850531007
```

Il se passe la même chose que pour l'exemple, l'ACP réduit les distances.

Utilisation de nouvelles variables créées, dont certaines à partir des variables textes

Jusqu'à présent, pour le modèle des plus proches voisins, on n'a utilisé que les variables numériques déjà présentes dans le dataset. On va maintenant tenter d'utiliser certaines variables textes en les transormant dans la mesure du possible en variables numériques.

In [43]:
```
reduced_data = PCA(n_components=2).fit_transform(dfNumeriquesSDSansDataManquantNormalise)
nbrs = NearestNeighbors(n_neighbors=6, algorithm="kd_tree").fit(reduced_data)
distances, indices = nbrs.kneighbors(reduced_data)
moyennes, moyenne = getMoy(distances)
print(moyennes)
print(moyenne)
```

```
[0.056165207206228168, 0.086340241817751934, 0.10810360366216899, 0.12424323504308683, 0.13940435075484608]
0.102851327697
```

La référence va donc être (ACP 2 + NN avec données initiales sans doublons sans données manquantes et normalisées):
[0.056165207206228168, 0.086340241817751934, 0.10810360366216899, 0.12424323504308683, 0.13940435075484608]
Moyenne générale : 0.102851327697
!!! Cela peut légèrement varier en fonction des tests, mais seulement de très peu (par exemple 0.102851 au lieu de 0.102849)

Si on ajoute comme variable le nombre de genres pour chaque film (on peut rechercher des films qui n'ont qu'un seul genre bien spécifique, ou des films très éclectiques), la distance n'est pas améliorée :

In [25]:
```
genres_num = dfOriginel['genres'].str.split('|').str.len()
dfGenresNum = pd.DataFrame({'genre_num' : genres_num})
dfOriginelGenresNum = pd.concat([dfOriginel, dfGenresNum], axis = 1)
getMoyennesNouvellesVariables(dfOriginelGenresNum)
```

```
[0.05704785870380142, 0.086622294019804327, 0.10853556229835186, 0.12476785836542936, 0.13992985864871302]

Moyenne générale des moyennes pour chaque ième film le plus proche : 0.103380686407
```

On peut également regarder le rendement d'un film (ratio recette / budget):

In [26]:
```
productif_num = dfOriginel['gross'] / dfOriginel['budget']
dfProductifNum = pd.DataFrame({'productif_num' : productif_num})
dfOriginelProductifNum = pd.concat([dfOriginel, dfProductifNum], axis = 1)
getMoyennesNouvellesVariables(dfOriginelProductifNum)
```

```
[0.056625301666866606, 0.086784551026396761, 0.10861160968618772, 0.1247509540724752, 0.13987531616280099]

Moyenne générale des moyennes pour chaque ième film le plus proche : 0.103329546523
```

On peut rechercher les similitudes dans celles des mots-clefs:

In [27]:
```
keywords_num = dfOriginel['plot_keywords'].str.split('|').str.len()
dKeywordsNum = {'keywords_num' : keywords_num}
dfKeywordsNum = pd.DataFrame(dKeywordsNum)
dKeywordsNumAug = pd.concat([dfOriginel, dfKeywordsNum], axis = 1)
getMoyennesNouvellesVariables(dKeywordsNumAug)
```

```
[0.056493078439378598, 0.0867207720040475, 0.10886322145541567, 0.12498983717455452, 0.14013021697452357]

Moyenne générale des moyennes pour chaque ième film le plus proche : 0.10343942521
```

On peut essayer avec l'ensemble de ces variables supplémentaires:

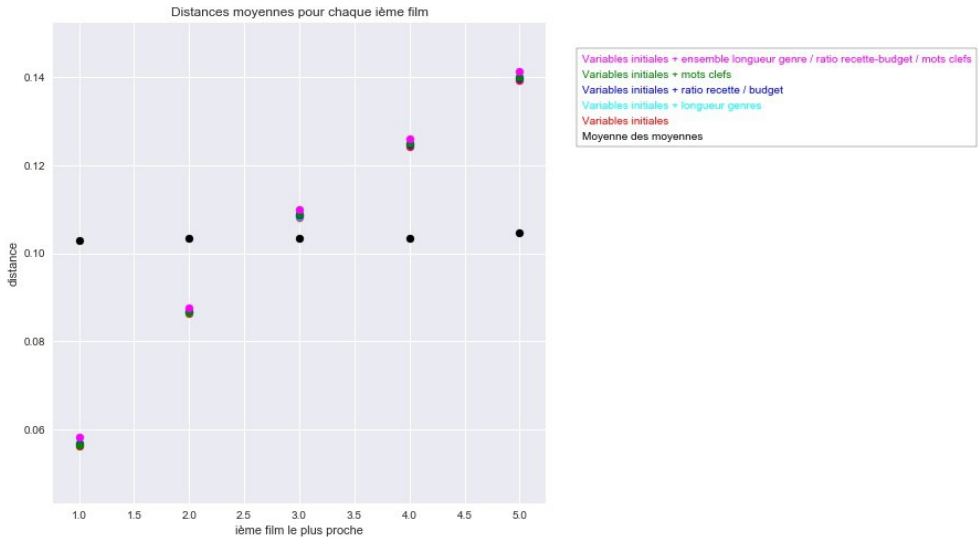
In [28]:
```
genres_num = dfOriginel['genres'].str.split('|').str.len()
productif_num = dfOriginel['gross'] / dfOriginel['budget']
keywords_num = dfOriginel['plot_keywords'].str.split('|').str.len()
d = {'genre_num' : genres_num, 'productif_num' : productif_num, 'keywords_num' : keywords_num}
df = pd.DataFrame(d)
dfAug = pd.concat([dfOriginel, df], axis = 1)
getMoyennesNouvellesVariables(dfAug)
```

```
[0.058325735013249509, 0.087729213375725534, 0.10987044934601664, 0.12602643817938505, 0.14147207613082574]

Moyenne générale des moyennes pour chaque ième film le plus proche : 0.104684782409
```

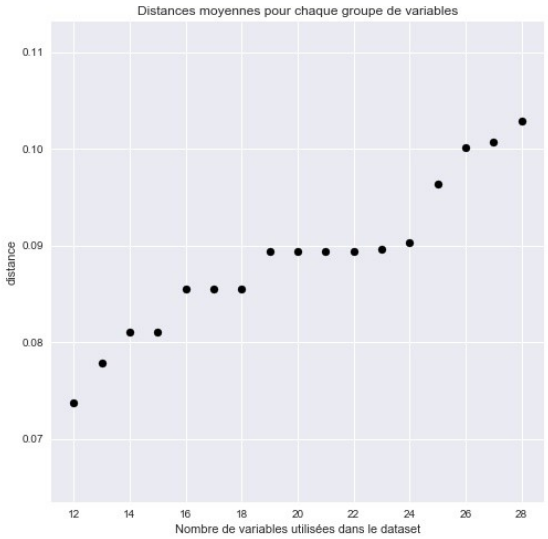
In [29]:
```
comparatifNouvellesVariables()
```

```
[0.10284969674, 0.10334206939225779, 0.10334822748040444, 0.10343857648208457, 0.10467563977853041]
```



Comme on peut le voir sur le graphique, aucun des modèles où on a ajouté des variables construites à partir des variables textes n'a amélioré les distances. Le meilleur reste le modèle initial (points rouges).

In [30]:
```
comparatifAjoutVariables()
```



La monotonie du graphe (les zones stables correspondent à l'ajout de variables textes qui n'entrent pas en jeu dans le calcul des distances pour le dataset initial) nous invite à nous demander s'il ne suffit pas de rajouter une variable pour que mécaniquement les distances soient augmentées puisque les distances ne semblent pas dépendre en premier lieu de la qualité des variables mais de leur quantité.

Ajout de variables en "one hot encoding" sur différentes variables texte :

In [31]:
```
cols = ['color', 'language', 'content_rating', 'country']
dfAug5 = pd.get_dummies(dfOriginel, columns = cols, prefix=['couleur', 'langage', 'age_auth', 'pays'])
getMoyennesNouvellesVariables(dfAug5)
```

```
[0.056180146688662731, 0.086313703524044841, 0.10810382896711757, 0.12424321638365141, 0.13938392166001781]

Moyenne générale des moyennes pour chaque ième film le plus proche : 0.102844963445
```

Commentaires :
Le one hot encoding a rajouté 156-28 = 128 variables binaires, cela a à première vue un peu amélioré le modèle mais de façon peu sensible, et lorsqu'on ne fait pas participer l'ACP dans la réduction des données, le modèle n'est pas amélioré (mêmes distances). On peut donc se poser la question de la réelle amélioration du modèle à l'aide de ces nouvelles variables et de la marge d'erreur des algorithmes sous-jacents.

Ajout de variables binaires d'appartenance à un genre particulier afin de voir s'il n'y a pas de rapprochements entre les films appartenant aux mêmes genres:

In [32]:
```
genre_df = dfOriginel['genres'].str.split('|')
liste_genres = []
for element in genre_df:
    for genre in element:
        if genre not in liste_genres:
            liste_genres.append(genre)
dfAug6 = dfOriginel.copy()
for genre in liste_genres:
    col_genre = np.where(dfOriginel['genres'].str.contains(genre) == True, 1, 0)
    label_genre = 'genre_'+genre
    d = {label_genre : col_genre}
    df_genre = pd.DataFrame(d)
    dfAug6 = pd.concat([dfAug6, df_genre], axis = 1)
getMoyennesNouvellesVariables(dfAug6)
```

```
[0.056181761535050445, 0.086307466275475156, 0.10809787160134711, 0.12423105818146189, 0.13937126641467848]

Moyenne générale des moyennes pour chaque ième film le plus proche : 0.102837884802
```

Le modèle n'est pas amélioré.

Ajout des variables numériques contenant les longueurs des variables textes aux différents noms d'acteur / réalisateur :

In [33]:
```
dfOriginelCopie = dfOriginel.copy()
director_name_long = dfOriginel['director_name'].str.len()
actor_1_name_long = dfOriginel['actor_1_name'].str.len()
actor_2_name_long = dfOriginel['actor_2_name'].str.len()
actor_3_name_long = dfOriginel['actor_3_name'].str.len()
dla = {'director_name_long' : director_name_long, 'actor_1_name_long' : actor_1_name_long, 'actor_2_name_long' : actor_2_name
_long, 'actor_3_name_long' : actor_3_name_long}
dfla = pd.DataFrame(dla)
dfAugla = pd.concat([dfOriginelCopie, dfla], axis = 1)
getMoyennesNouvellesVariables(dfAugla)
```

```
[0.056297935227623906, 0.086608344613766791, 0.10850503298544442, 0.12482044753188518, 0.13993856038465838]

Moyenne générale des moyennes pour chaque ième film le plus proche : 0.103234064149
```

Le modèle n'est également pas amélioré.

Commentaires :

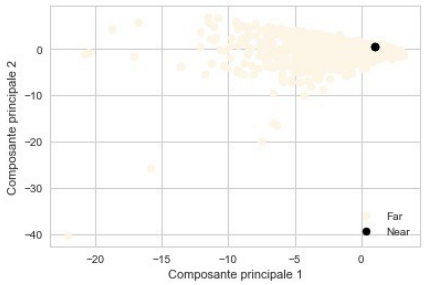
- Aucune des variables ajoutées n'améliore le modèle, semblant ajouter "mécaniquement" de la distance à chaque nouvelle variable.
- On peut alors se demander si comparer les distances pour des ensembles différents de variables a réellement un sens. Que pour un ensemble donné de variables, on compare les distances des films à l'intérieur de cet ensemble, cela peut paraître convenable puisque le cadre de comparaison est le même, mais dans ce contexte non supervisé, modifier le nombre de dimension du problème crée autant de modèles possibles que de choix de variables.

Positionnement des 6 films sur le graphe de l'ACP

In [34]:
```
# les 5 plus proches films du 4769, "20 000 lieues sous les mers" qu'on a traité précédemment
graphePositionProchesDansACP(dfNumeriquesSDSansDataManquantNormalise, dfOriginelSansDoublon, 4769)
```

```
[4769 3092 4374 3386 1337 1983]

<matplotlib.figure.Figure at 0x10a6af28>
```



Pour ce film, ses plus proches voisins sont si rapprochés qu'on ne les discerne pas sur l'ACP.

A fins de comparaisons, on peut regarder deux familles de films "opposés" sur l'ACP, le film d'indice 1026 (Miss Congeniality) et le 3 (The Dark Knight Rises):

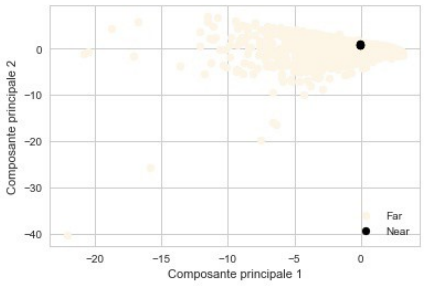
In [35]:
```
print(dfOriginelSansDoublon.iloc[1026,11])
print(dfOriginelSansDoublon.iloc[3,11])
```

```
Miss Congeniality
The Dark Knight Rises
```

Miss Congeniality et ses proches se situent sur l'ACP quasiment à 0 pour leurs deux premières composantes principales, et The Dark Knight Rises et ses proches sont bien plus dispersés et ont une valeur élevée pour la première composante. On peut donc tenter d'interpréter cela au niveau des valeurs des variables des composantes pour ces films:

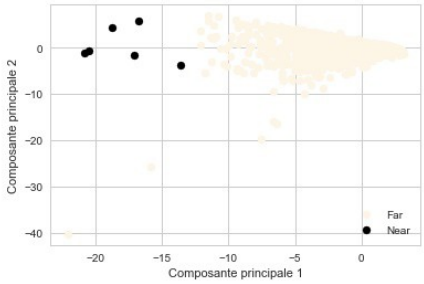
In [36]:
```
graphePositionProchesDansACP(dfNumeriquesSDSansDataManquantNormalise, dfOriginelSansDoublon, 1026)
```

```
[1026 2282 2063  921 2658 3571]
<matplotlib.figure.Figure at 0x104df9e8>
```



In [37]:
```
graphePositionProchesDansACP(dfNumeriquesSDSansDataManquantNormalise, dfOriginelSansDoublon, 3)
```

```
[  3  86 751  57   7  85]
<matplotlib.figure.Figure at 0x10b34860>
```



La méthode positions renvoie les positions de ces films pour les 2 premières variables (num_voted_users et num_critic_for_reviews) de la CP1, la première variable cast_total_facebook_like de la CP2 et une variable, director_facebook_likes, qui n'apparaît dans aucune des deux premières composantes principales.

Pour cette série de films les plus proches du 1026 (Miss Congeniality), les valeurs sur les CP1 et CP2 sont proches de 0.

In [38]:
```
positions([1026, 2282, 2063, 921, 2658, 3571])
```

```
num_voted_users
874
1235
1453
1718
1898
1429
num_critic_for_reviews
721
228
483
658
923
272
cast_total_facebook_like
3536
2738
2491
2754
3481
4507
director_facebook_likes
98
110
125
840
840
315
```

On peut voir que les positions de ces films pour chaque variable sont, dans une certaine mesure, pas trop éloignées (souvent dans le même millier), ce qui dénote une certaine ressemblance pour ces films. Par contre, ce ne sont pas des classements très élevés, ces variables ne semblent pas, pour ces films, avoir de fortes valeurs.

Il n'en va pas de même pour les proches du film 3 ci-dessous (The Dark Knight Rises) dont on voit sur le graphe qu'ils ont des valeurs élevées sur la composante principale 1 de l'ACP. On a bien, sur les deux premières variables de la CP1, des classements élevés (dans les 20 premiers sur près de 5000) pour ces films, et un peu moins pour la première variable de la CP2 (classements ci-dessous).
Il est à noter que le réalisateur Christopher Nolan apparaît 4 fois sur les 6 films les plus proches du sombre chevalier.

In [39]:
```
positions([3, 86, 751, 57, 7, 85])
```

```
num_voted_users
11
3
14
2
134
18
num_critic_for_reviews
1
14
8
13
23
7
cast_total_facebook_like
11
24
20
63
15
343
director_facebook_likes
1
1
125
1
125
1
```

6- Présentation de 3 modèles possibles, résultats pour le film d'indice 0, Avatar

A chaque choix d'un ensemble de variables (initiales ou construites) semble correspondre un modèle qui permet d'obtenir, dans le cadre particulier de ces variables, 5 films les plus proches. Le premier modèle utilise toutes les variables numériques initiales. Les deux autres tentent de rajouter du sens en se rapprochant plus des goûts des gens.

- Modèle avec toutes les variables

In [40]:
```
modeleAllVar(dfOriginelSansDoublon)
```

```
Film         0                                                Avatar
             1859                                The Shawshank Redemption
             247     The Lord of the Rings: The Fellowship of the Ring
             9                         Batman v Superman: Dawn of Justice
             306          The Lord of the Rings: The Return of the King
             642                                             Fight Club
réalisateur  0                                          James Cameron
             1859                                      Frank Darabont
             247                                        Peter Jackson
             9                                            Zack Snyder
             306                                        Peter Jackson
             642                                        David Fincher
acteur_1     0                                            CCH Pounder
             1859                                      Morgan Freeman
             247                                      Christopher Lee
             9                                           Henry Cavill
             306                                        Orlando Bloom
             642                                            Brad Pitt
genres       0                          Action|Adventure|Fantasy|Sci-Fi
             1859                                        Crime|Drama
             247                         Action|Adventure|Drama|Fantasy
             9                                  Action|Adventure|Sci-Fi
             306                         Action|Adventure|Drama|Fantasy
             642                                               Drama
dtype: object
```

Commentaires :

- Avantage : ce modèle semble stable, lorsqu'on le relance plusieurs fois, on obtient toujours les mêmes résultats.
- Désavantage : Le fait d'utiliser toutes les variables fait perdre du sens aux résultats car toutes les variables ont alors la même importance, ce qui est rarement le cas lorqu'une personne souhaite connaître les films les plus proches d'un film donné. On le voit d'ailleurs dans cet exemple où un "Crime|Drama" est le plus proche d'un "Action|Adventure|Fantasy|Sci-Fi", et où le 5ème plus proche d'Avatar est.. Fight Club, deux films qui à première vue n'ont pas grand chose en commun.

On peut alors essayer de donner du sens en se mettant à la place d'un utilisateur, qui par exemple choisirait un film de Fantasy-sci-fi et souhaiterait ne voir que des films de cette catégorie de films. On peut alors prendre un modèle qui ne concerne que les genres:

- Modèle avec uniquement les genres

In [41]:
```
modeleVarGenres(dfOriginelSansDoublon)
```

```
Film         0                                       Avatar
             1468    Star Wars: Episode VI - Return of the Jedi
             219      Star Wars: Episode I - The Phantom Menace
             480          The League of Extraordinary Gentlemen
             3515     Beastmaster 2: Through the Portal of Time
             2583                             Highlander: Endgame
réalisateur  0                                   James Cameron
             1468                              Richard Marquand
             219                                   George Lucas
             480                               Stephen Norrington
             3515                                 Sylvio Tabet
             2583                             Douglas Aarniokoski
acteur_1     0                                    CCH Pounder
             1468                                Harrison Ford
             219                                Natalie Portman
             480                                  Jason Flemyng
             3515                              Michael Berryman
             2583                             Christopher Lambert
genres       0                 Action|Adventure|Fantasy|Sci-Fi
             1468              Action|Adventure|Fantasy|Sci-Fi
             219               Action|Adventure|Fantasy|Sci-Fi
             480               Action|Adventure|Fantasy|Sci-Fi
             3515              Action|Adventure|Fantasy|Sci-Fi
             2583              Action|Adventure|Fantasy|Sci-Fi
dtype: object
```

Commentaires :

- Avantage : ce modèle gagne du sens pour l'utilisateur car il est plus proche de la façon dont l'utilisateur va choisir ses films que le modèle où toutes les variables sont utilisées. On retrouve bien parmi les 5 films les plus proches d'Avatar, 5 films de la même catégorie de genres "Action|Adventure|Fantasy|Sci-Fi"
- Désavantage : il n'est pas stable, quand on le relance plusieurs fois sur le même film, il donne des résultats différents.

Une solution serait peut-être de :

- restreindre le dataset aux seuls films partageant le genre du film désiré
- relancer l'algorithme des 5 plus proches voisins à ce sous-dataset et à toutes les autres variables comme pour le premier modèle. On aurait alors un modèle stable qui donnerait toujours les mêmes résultats et qui aurait en plus un peu de sens.

On peut également envisager un modèle avec en plus le nom du réalisateur et celui de principal acteur qui sont souvent des critères essentiels de choix de films.

- Modèle avec les variables 'genres', 'director_name' et 'actor_1_name'

In [42]:
```
modeleVarGenresReaActeur1(dfOriginelSansDoublon)
```

```
Film         0                                       Avatar
             2583                             Highlander: Endgame
             4052                                Silver Medallist
             2379                                      Red Sonja
             144                                       Star Trek
             215     Star Wars: Episode III - Revenge of the Sith
réalisateur  0                                     James Cameron
             2583                             Douglas Aarniokoski
             4052                                        Hao Ning
             2379                               Richard Fleischer
             144                                      J.J. Abrams
             215                                   George Lucas
acteur_1     0                                    CCH Pounder
             2583                             Christopher Lambert
             4052                                        Jack Kao
             2379                                Ernie Reyes Jr.
             144                                 Chris Hemsworth
             215                                 Natalie Portman
genres       0                 Action|Adventure|Fantasy|Sci-Fi
             2583              Action|Adventure|Fantasy|Sci-Fi
             4052                     Action|Adventure|Comedy
             2379                    Action|Adventure|Fantasy
             144                      Action|Adventure|Sci-Fi
             215               Action|Adventure|Fantasy|Sci-Fi
dtype: object
```

Commentaires :

- Avantage/désavantage : les mêmes que le précédent, il est en position intermédiaire entre celui avec toutes les variables et celui avec une seule variable (initiale).

Résultats

Par exemple, pour le premier de ces modèles (toutes variables), la commande : http://[IP_serveur]:[port]/recommend/0
a donné ceci pour le film d'identifiant (d'indice) 0 (c'est-à-dire le film Avatar, le premier de la liste) :

```
{
  "_results": [
    {
      "id": 0,
      "name": "Avatar"
    },
    {
      "id": 1859,
      "name": "The Shawshank Redemption"
    },
    {
      "id": 247,
      "name": "The Lord of the Rings: The Fellowship of the Ring"
    },
    {
      "id": 9,
      "name": "Batman v Superman: Dawn of Justice"
    },
    {
      "id": 306,
      "name": "The Lord of the Rings: The Return of the King"
    },
    {
      "id": 642,
      "name": "Fight Club"
    }
   ]
}
```


6- Conclusions

1- L'ACP ne permet pas de faire ressortir du dataset des orientations très marquées de films. Il y a une très grande diversité de films mais aucun groupe particulier ne semble être mis en exergue.
2- La méthode des plus proches voisins permet d'obtenir pour un film donné, les 5 films qui lui sont les plus proches. Si on n'utilise pas d'ACP et qu'on utilise la méthode des plus proches voisins sur le dataset, alors on ne donne pas de préférence à une variable particulière, chaque variable a le même poids, ce qui produit des films les plus proches sans réelles similarités apparentes.
3- Ajouter des variables n'améliore pas le modèle initial des 5 plus proches films basé sur le dataset initial. Pour chaque ajout de variable, la distance entre les films augmente. Mais réciproquement, en diminuant le nombre de variables, on diminue également la distance moyenne entre les films.
4- Il est également possible de réduire les distances en effectuant une ACP, on obtient alors une certaine orientation de la famille des films les plus proches en fonction de leur projection sur les composantes principales de l'ACP. L'interprétation ne se fait plus suivant toutes les variables prises de façon uniforme (même importance pour chacune), mais suivant les principales variables des principales composantes principales. En ce qui concerne le nombre de composantes principales à prendre en compte, il semble préférable de n'utiliser que les deux premières composantes principales puisque lorsqu'on augmente le nombre de composantes, la distance entre les films augmente.
5- L'utilisation de l'ACP, qui permet de réduire les distances et donc d'améliorer le modèle (si on considère que le critère de comparaison des modèles est la distance entre les films), est orientée par le jeu de donnée total. Les 5 plus proches voisins d'un film donné sont donc le résultat d'une tendance globale de l'ensemble des utilisateurs, tendance qui concerne l'ensemble des variables. C'est donc en quelque sorte la population totale du dataset qui donne ses préférences, mais il n'en ressort aucun réel critère de préférence.
Il est alors probable qu'en l'état actuel des informations du dataset, une personne qui choisirait un film et à qui on donnerait les 5 films les plus proches suivant cette méthode aurait du mal à trouver de réelles ressemblances entre ces films. Et il semble que cela soit effectivement le cas.
6- On peut essayer de donner un peu de sens aux résultats en choisissant un/des modèle/s basé/s sur un choix de variables particulières reflétant les recherches habituelles des gens en terme de films proches. Il est en effet assez courant qu'une personne souhaite connaître tous les films d'un certain genre proches d'un film qu'elle a adoré, ou tous les films d'un réalisateur proches du réalisateur qu'elle vient de découvrir. Il en va de même pour un acteur. On retrouve d'ailleurs dans la plupart des points de vente de films, des catégories "réalisateur", "acteur", "genre" etc. 3 modèles ont donc été proposés, le modèle avec toutes les variables initiales, celui avec uniquement les genres, et celui avec les genres, le nom du réalisateur et le nom de l'acteur principal.
7- Deux de ces modèles ont été implémentés via services REST sur AWS (Amazon Web Service)

Améliorations possibles : On pourrait sans doute améliorer le modèle en intégrant des informations sur les critères de choix de films par les utilisateurs, afin d'étudier statistiquement leurs préférences, leurs hiérarchies de goûts. En ajoutant des données utilisateurs, on pourrait sans doute améliorer les distances en déterminant quelles sont les variables les plus intéressantes et donner un peu plus de sens aux 5 plus proches films, c'est-à-dire se rapprocher des goûts des utilisateurs particuliers plutôt que de la tendance générale de l'ensemble de la population.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
import seaborn as sns
%matplotlib inline
import math
from matplotlib.gridspec import GridSpec
from sklearn.decomposition import PCA
import sklearn.preprocessing as prep
from sklearn.neighbors import NearestNeighbors
import matplotlib.artist as artists
from matplotlib.offsetbox import AnchoredOffsetbox, TextArea, HPacker, VPacker
from os import path
from scipy.misc import imread
import matplotlib.pyplot as plt
import random
from wordcloud import WordCloud, STOPWORDS
from sklearn.feature_extraction.text import CountVectorizer
```

```
fichier = 'C:/Users/gewurz14/.spyder/notebooks/projet3/movie_metadata.csv'

dfOriginel = pd.read_csv(fichier, sep=',') # dataframe des données initiales

dfOriginelSansDoublon = dfOriginel.drop_duplicates(subset='movie_title', keep='last')

def proportionDonneesManquantes(_df):
    plt.figure(figsize=(10, 6))
    _df.isnull().mean(axis=0).plot.barh()
    plt.title(u"Proportion de données manquantes pour chaque variable")



def make_autopct(values):
    def my_autopct(pct):
        """Customisation de l'affichage des valeurs {p:.2f}% ({v:d})
        Arguments:
        values -- nombres de variables pour chaque section
        pct -- pourcentages de variables pour chaque section
        """
        total = sum(values)
        val = int(round(pct*total/100.0))
        if val == 0:
            return ''
        else:
            return '{p:.2f}% ({v:d})'.format(p=pct, v=val)
    return my_autopct

def make_autopct2(values):
    def my_autopct2(pct):
        """Customisation de l'affichage des valeurs {v:d}
        Arguments:
        values -- nombres de variables pour chaque section
        pct -- pourcentages de variables pour chaque section
        """
        total = sum(values)
        val = int(round(pct*total/100.0))
        return '{v:d}'.format(v=val)
    return my_autopct2

def graphCamembertRemplissage(_df):
    nb_enreg = _df.shape[0]
    dfOriginelSansDoublonDataManquant = 100*(1-_df.count()/_df.shape[0])
    classement=[]
    for i in range(10):
        classement.append(0)

    for i in range (_df.count().shape[0]):
        classement[int(math.ceil(_df.count()[i]/(nb_enreg/10.0))-1)] += 1

    explode = (0.10, 0, 0, 0.1, 0.25, 0.1, 0, 0, 0, 0)

    the_grid = GridSpec(2,2)
    plt.clf()
    plt.figure(figsize=(17,10))
    plt.subplot(the_grid[:,0], aspect=1)
    _, _, autotexts = plt.pie(classement, explode=explode, autopct=make_autopct(classement), startangle=90, shadow=True, pctd
istance=0.5) #'%1.1f%%'
    plt.title(u"% de variables numériques (nombre de variables) ayant\n entre . et . % de données renseignées (couleurs)", bb
ox={'facecolor':'0.8', 'pad':5})
    autotexts[8].set_color('white')
    autotexts[8].set_fontsize(16)
    autotexts[9].set_color('white')
    autotexts[9].set_fontsize(16)
    plt.axis('equal')
    labels = ['0-10%', '10-20%', '20-30%', '30-40%', '40-50%', '50-60%', '60-70%', '70-80%', '80-90%', '90-100%']
    plt.legend(labels, loc='lower right')

    nbVarMoins0virg5pcDataManquant = dfOriginelSansDoublonDataManquant[dfOriginelSansDoublonDataManquant < 0.5].count()
    nbVarPlus0virg5pcDataManquant = dfOriginelSansDoublonDataManquant[dfOriginelSansDoublonDataManquant >= 0.5].count()
    values = [nbVarMoins0virg5pcDataManquant, nbVarPlus0virg5pcDataManquant]
    plt.subplot(the_grid[0,1], aspect=1)
    _, _, autotexts = plt.pie(values, autopct=make_autopct2(values))
    autotexts[0].set_color('white')
    autotexts[1].set_color('white')
    for i in range(2):
        autotexts[i].set_fontsize(16)
    plt.title(u"Nombre de variables ayant\n moins de 0.5 % de données manquantes", bbox={'facecolor':'0.8', 'pad':5})
    plt.legend(['<0.5% manquantes', '>=0.5% manquantes'], loc='lower right')
    plt.subplot(the_grid[1,1], aspect=1)
    nbVarMoins5pcDataManquant = dfOriginelSansDoublonDataManquant[dfOriginelSansDoublonDataManquant < 5].count()
    nbVarPlus5pcDataManquant = dfOriginelSansDoublonDataManquant[dfOriginelSansDoublonDataManquant >= 5].count()
    values = [nbVarMoins5pcDataManquant, nbVarPlus5pcDataManquant]
    _, _, autotexts = plt.pie(values, autopct=make_autopct2(values))
    autotexts[0].set_color('white')
    autotexts[1].set_color('black')
    for i in range(2):
        autotexts[i].set_fontsize(16)
    plt.title(u"Nombre de variables ayant\n moins de 5 % de données manquantes", bbox={'facecolor':'0.8', 'pad':5})
    plt.legend(['<5% manquantes', '>5% manquantes'], loc='lower right')
    plt.show()
```

```
def getDfNumerique(_df):
    """Sépare les variables numériques et non numériques
    Arguments:
    _df -- dataframe contenant les données initiales
    Retour:
    dfNumeriques -- dataframe contenant les variables numériques
    dfNonNumeriques -- dataframe contenant les variables non numériques
    """

    # séparation variables numériques / non numériques
    dfNumeriques = pd.DataFrame()
    dfNonNumeriques = pd.DataFrame()

    dfNumeriques = _df.select_dtypes(include = ['float64', 'int64']).iloc[:, :]
    dfNonNumeriques = _df.select_dtypes(exclude = ['float64', 'int64']).iloc[:, :]

    return dfNumeriques, dfNonNumeriques



def grapheDonneesManquantesVarNumNonNum(_df,  _dfNumeriques, _dfNonNumeriques):

    dfNumeriquesSansDoublonDataManquantes = 100*(1-_dfNumeriques.count()/_dfNumeriques.shape[0])
    dfNonNumeriquesSansDoublonDataManquantes = 100*(1-_dfNonNumeriques.count()/_dfNonNumeriques.shape[0])

    plt.figure(figsize=(12,4))

    plt.subplot(121)

    X1 = (100-dfNumeriquesSansDoublonDataManquantes).sort_values()
    plt.bar(range(len(X1)), X1)
    plt.xlabel("indice des variables")
    plt.ylabel("%")
    plt.title(u'% ordonné de remplissage des variables numériques')

    plt.subplot(122)

    X2 = (100-dfNonNumeriquesSansDoublonDataManquantes).sort_values()
    plt.bar(range(len(X2)), X2)
    plt.xlabel("indice des variables")
    plt.ylabel("%")
    plt.title(u'% ordonné de remplissage des variables non numériques')

def graphesExtremes(_coeff, _dfNumeriques):
    """Graphes des % de valeurs extrêmes
    Arguments:
    _coeff -- coefficient multiplicateur pour déterminer le seuil (moyenne + coeff * écart-type)
    """
    valExtr = _dfNumeriques[_dfNumeriques>_dfNumeriques.mean()+_coeff*_dfNumeriques.std()].count()
    liste_nom_val_extreme = []
    valCo = _dfNumeriques.count()
    for nom in _dfNumeriques.columns:
        if valExtr[nom] > 0:
            liste_nom_val_extreme.append((nom, 100.0/valCo[nom]*valExtr[nom]))

    liste_nom_val_extreme_sorted = sorted(liste_nom_val_extreme, key=lambda x:x[1])
    liste_val_extreme_sorted = []
    for i in range(len(liste_nom_val_extreme_sorted)):
        liste_val_extreme_sorted.append(liste_nom_val_extreme_sorted[i][1])

    return liste_val_extreme_sorted

def deuxGraphesExtremes(_df, _dfNumeriques):

    liste_val_extreme_sorted15 = graphesExtremes(1.5, _dfNumeriques)
    liste_val_extreme_sorted3 = graphesExtremes(3, _dfNumeriques)

    plt.clf()
    plt.figure(figsize=(15,4))
    plt.subplot(121)
    plt.bar(range(len(liste_val_extreme_sorted15)), liste_val_extreme_sorted15)
    plt.xlabel("indice des variables")
    plt.ylabel("%")
    plt.title(u'% ordonné du nombre de valeurs extrêmes (> mean + 1.5std)')
    plt.subplot(122)
    plt.xlabel("indice des variables")
    plt.ylabel("%")
    plt.bar(range(len(liste_val_extreme_sorted3)), liste_val_extreme_sorted3)
    plt.title(u'% ordonné du nombre de valeurs extrêmes (> mean + 3std)')
    plt.show()
def dfBoxPlot(_df, _title):
    """Dessine le boxplot
    Arguments:
    _df -- dataframe contenant les données initiales
    _title -- titre du boxplot
    """
    plt.figure(figsize=(12,6))
    colors=dict(boxes='DarkGreen', whiskers='DarkOrange', medians='DarkBlue', caps='Gray')
    boxprops = dict(linewidth=2)
    medianprops = dict(linewidth=2)
    meanpointprops = dict(marker='D', markeredgecolor='black', markerfacecolor='firebrick')
    flierprops = dict(marker='o', linestyle='none', markerfacecolor='green', markersize=6)
    ax = _df.plot.box(showmeans=True, showfliers=True, flierprops=flierprops, boxprops=boxprops, medianprops=medianprops, col
or=colors, meanprops=meanpointprops)
    ax.set_ylabel('Valeurs des variables')
    ax.set_title(_title)
```

```python
def heatmap(_df, _methode):
    """Affichage de la heatmap de la matrice de corrélation des variables continues
    Arguments:
    _df -- dataframe contenant les variables
    _methode -- méthode statistique pour la corrélation, Pearson, Spearman ou Kendall
    """

    # suppression des figures
    plt.clf()
    # on calcule la matrice de corrélation pour le dataframe _df passé en paramètre
    corr = _df.corr(method=_methode)
    # masque d'affichage de la heatmap
    mask = np.zeros_like(corr, dtype=np.bool)
    mask[np.triu_indices_from(mask)] = True
    f, ax = plt.subplots(figsize=(12,12))
    cmap = sns.diverging_palette(220, 10, as_cmap=True)

    # traçage de la heatmap
    sns.heatmap(corr, cmap=cmap, cbar_kws={"shrink": .5}, mask=mask, annot=True)
    plt.show()

def composantesPrincipale(_dfNormalise, _dfNonNormalise, _nombre_composantes, _numero_composante):
    """Affichage des dix premières variables importantes d'une composante principale de l'ACP
    Arguments:
    _dfNormalise -- dataframe contenant les variables normalisées pour l'ACP
    _dfNonNormalise -- dataframe contenant les variables non normalisées
    _nombre_composante -- nombre de composantes principales pour l'ACP
    _numero_composante -- numéro de la composante principale à afficher
    """
    pca = PCA(n_components=_nombre_composantes).fit(prep.scale(_dfNormalise))

    pca_df = pd.DataFrame(pca.components_[_numero_composante])
    pca_df.columns = ['coeff']

    feat_pca = pd.DataFrame(_dfNonNormalise.columns.values)
    feat_pca.columns = ['features']

    pca_feat_importances = pd.merge(feat_pca, pca_df, left_index=True, right_index=True)
    pca_feat_importances['importances']= pca_feat_importances['coeff']/pca_feat_importances['coeff'].sum()*100
    pca_feat_importances['importances'] = pca_feat_importances['importances'].apply(lambda x:abs(int(x)))
    print(pca_feat_importances.sort(['importances'], ascending=[0]).head(10))

def getPropre(_df, _graphe):
    """Méthode permettant d'obtenir les valeurs propres et les vecteurs propres de la matrice de corrélation, affiche égaleme
nt le graphe de la participation de chaque variable à l'explication de la variance
    Arguments:
    _df -- dataframe contenant les variables
    _graphe -- booléen indiquant si on veut que le graphe soit affiché ou non
    Retour: la liste des tuples (valeurs propres, vecteurs propres)
    """

    cor_mat1 = np.corrcoef(_df.T)
    eig_vals, eig_vecs = np.linalg.eig(cor_mat1)
    tot = sum(eig_vals)
    var_exp = [(i / tot)*100 for i in sorted(eig_vals, reverse=True)]
    cum_var_exp = np.cumsum(var_exp)
    if _graphe:
        with plt.style.context('seaborn-whitegrid'):
            plt.clf()
            plt.figure(figsize=(6, 4))
            plt.bar(range(_df.shape[1]), var_exp, alpha=0.5, align='center',
                    label='Variance expliquee individuellement')
            plt.step(range(_df.shape[1]), cum_var_exp, where='mid',
                     label='Variance expliquee par cumul')
            plt.ylabel('Pourcentage de variance expliquee')
            plt.xlabel('Composantes principales')
            plt.legend(loc='best')
            plt.tight_layout()
            plt.show()

    # Make a list of (eigenvalue, eigenvector) tuples
    eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in range(len(eig_vals))]

    return eig_pairs

def grapheACP2Composantes(_val_vect_propres, _df, _var_prediction):
    """Représentation graphique des deux composantes principales de l'ACP
    Arguments:
    _val_vect_propres -- tuples des valeurs propres et des vecteurs propres de la matrice de corrélation
    _df -- dataframe des variables continues dont les valeurs manquantes ont été mises à 0
    _var_prediction -- variable à prédire
    """

    # tri des tuples (eigenvalue, eigenvector) par ordre décroissant
    _val_vect_propres.sort(key=lambda x: x[0], reverse=True)

    matrix_w = np.hstack((_val_vect_propres[0][1].reshape(_df.shape[1],1), _val_vect_propres[1][1].reshape(_df.shape[1],1)))

    Y = _df.dot(matrix_w)

    with plt.style.context('seaborn-whitegrid'):
        plt.clf()
        plt.figure(figsize=(6, 4))
        for lab, col in zip(('B', 'M'), ('blue', 'red')):
            plt.scatter(Y[_var_prediction==lab, 0], Y[_var_prediction==lab, 1], label=lab, c=col)
        #plt.scatter(Y[:,0],Y[:,1], c='goldenrod',alpha=0.5)
        plt.xlabel('Composante principale 1')
        plt.ylabel('Composante principale 2')
        plt.legend(loc='lower right')
        plt.show()
```

```
def voisinsProches(_nb_voisins, _algorithm, _df, _num_film):
    nbrs = NearestNeighbors(n_neighbors=_nb_voisins, algorithm=_algorithm).fit(_df)
    distances, indices = nbrs.kneighbors(_df)
    print(indices[_num_film])
    print(distances[_num_film])

def getFilmNouvelIndice(_ind_film):
    for i in range(dfOriginelSansDoublon.shape[0]):
        if dfOriginelSansDoublon.iloc[i,11] == dfOriginel.iloc[_ind_film,11]:
            break
    return i

def graphesDistACP(tab_dist):
    X = np.linspace(1,6,6)
    f, ax = plt.subplots(figsize=(8,8))
    text0 = TextArea("sans ACP", textprops=dict(color="black"))
    text1 = TextArea("ACP 2", textprops=dict(color="red"))
    text2 = TextArea("ACP 3", textprops=dict(color="blue"))
    text3 = TextArea("ACP 4", textprops=dict(color="green"))
    text4 = TextArea("ACP 5", textprops=dict(color="magenta"))
    box = VPacker(children=[text4, text3, text2, text1, text0], pad=5, sep=5)
    anchored_box = AnchoredOffsetbox(loc=3, child=box, pad=0., bbox_to_anchor=(1.05, 0.73), bbox_transform=ax.transAxes,)
    ax.add_artist(anchored_box)

    plt.scatter(X, tab_dist[0], color='black')
    plt.scatter(X, tab_dist[1], color='red')
    plt.scatter(X, tab_dist[2], color='blue')
    plt.scatter(X, tab_dist[3], color='green')
    plt.scatter(X, tab_dist[4], color='magenta')
    ax.set_title(u'Distances pour chaque ième film')
    ax.set_xlabel(u'ième film le plus proche')
    ax.set_ylabel('distance')
def score(_algo, _dfNumeriquesSDSansDataManquantNormalise):
    nbrs = NearestNeighbors(n_neighbors=6, algorithm=_algo).fit(_dfNumeriquesSDSansDataManquantNormalise)
    distances, indices = nbrs.kneighbors(_dfNumeriquesSDSansDataManquantNormalise)
    return distances, indices

def getMoy(distances):
    moyennes_distances = []
    for i in range(1,6):
        moyennes_distances.append(np.mean(distances[:,i]))
    return moyennes_distances, np.mean(moyennes_distances)

def getMoyMoyACP2(_dfNumeriquesSDSansDataManquantNormalise, _method):
    distances, indices = score(_method, dfNumeriquesSDSansDataManquantNormalise)
    moyennes, moyenne = getMoy(distances)
    print("Sans ACP, moyenne des distances pour les 5 films les plus proches : \n{0}".format(moyennes))
    print("Sans ACP, moyenne des moyennes des distances des 5 films les plus proches : {0}\n".format(moyenne))

    reduced_data = PCA(n_components=2).fit_transform(dfNumeriquesSDSansDataManquantNormalise)
    nbrs = NearestNeighbors(n_neighbors=6, algorithm=_method).fit(reduced_data)
    distances, indices = nbrs.kneighbors(reduced_data)
    moyennes, moyenne = getMoy(distances)
    print("Avec ACP 2 composantes, moyenne des distances pour les 5 films les plus proches : \n{0}".format(moyennes))
    print("Avec ACP 2 composantes, moyenne des moyennes des distances des 5 films les plus proches : {0}".format(moyenne))

def distinctGenres(row):
    genres = np.array(row['genres'].split('|'))
    for genre in genres:
        if genre not in liste_genre:
            liste_genre.append(genre)
global liste_genre
liste_genre = []
global d
d = {}
def genreRemap1(row):
    genres = np.array(row['genres'].split('|'))
    for genre in genres:
        d[genre]['year'] =  d[genre]['year'] + row['title_year']
        d[genre]['imdb_score'] =  d[genre]['imdb_score'] + row['imdb_score']
        d[genre]['budget'] =  d[genre]['budget'] + row['budget']
        d[genre]['gross'] =  d[genre]['gross'] + row['gross']
        d[genre]['nb_occur'] = d[genre]['nb_occur'] + 1
def getDfGenre(_df):
    # On ne prend que les enregistrements où on a toutes les valeurs de ces variables
    df_clean = _df[['budget', 'genres', 'title_year', 'imdb_score', 'gross']].dropna()
    df_clean.apply(distinctGenres, axis = 1)
    for genre in liste_genre:
        d[genre] = {'year': 0, 'imdb_score': 0, 'budget': 0, 'nb_occur': 0, 'gross': 0}
    df_clean.apply(genreRemap1, axis = 1)
    index = []
    year = []
    budget = []
    score = []
    nb_occur = []
    genres = []
    gross = []
    for genre in d:
        index.append(genre)
        if d[genre]['nb_occur'] > 0:
            year.append(d[genre]['year']/d[genre]['nb_occur'])
            budget.append(d[genre]['budget']/d[genre]['nb_occur'])
            score.append(d[genre]['imdb_score']/d[genre]['nb_occur'])
            gross.append(d[genre]['gross']/d[genre]['nb_occur'])
            nb_occur.append(d[genre]['nb_occur'])
    d2 = {'year' : year, 'budget' : budget, 'nb_occur' : nb_occur, 'score' : score, 'genre' : index, 'gross' : gross}
    dfGenreTot = pd.DataFrame(d2, index=index)
    return dfGenreTot, year, index, df_clean
```

```
def grapheNombreFilmParGenre(_df):
    f, ax = plt.subplots(figsize = (8, 5))
    sns.barplot(x = 'nb_occur', y = 'genre', data = _df)
    ax.set_title('Nombre de films par genre')
    ax.set_xlabel('nombre')
    ax.set_ylabel('genre')

def grapheScoreFilmParGenre(_df):
    f, ax = plt.subplots(figsize = (20, 5))
    sns.barplot(x = 'genre', y = 'score', data = _df)
    ax.set_title('Score IMDB moyen par genre')
    ax.set_xlabel('genre')
    ax.set_ylabel('score')

def grapheAnneeMoyenneFilmParGenre(year, index):
    f, ax = plt.subplots(figsize = (8, 5))
    colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k', 'lightblue', 'b', 'g', 'r', 'c', 'm', 'y', 'k', 'lightblue', 'b', 'g', 'r',
'c', 'm', 'y', 'k', 'lightblue']
    markers = ['o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', '*', '*', '*', '*', '*', '*', '*', '*', 'v', 'v', 'v', 'v', 'v', 'v',
'v', 'v']
    for x, annee, color, label, marker in zip(range(len(year)), year, colors, index, markers):
        ax.plot(x, annee, color=color, marker=marker, markersize=10, label=label)
    ax.legend(loc='center left', bbox_to_anchor=(1,0.5))
    ax.set_title(u'Année moyenne de sortie des films par genre')

def calculDistIndACPNN(_algo, _df):
    """
    Méthode qui calcule les 5 plus proches voisins de tous les films présents dans le dataframe _df
    Arguments:
    _algo -- l'algorithme utilisé par la méthode NearestNeighbors
    _df -- dataframe contenant les films
    Retour:
    distances tableau des 5 distances pour les 5 films les plus proches
    indices tableau des 5 indices représentant les 5 films les plus proches
    """
    dfSansDoublon = _df.drop_duplicates(subset='movie_title', keep='last')
    dfNumeriquesSD, _ = getDfNumerique(dfSansDoublon)
    dfNumeriquesSDSansDataManquant = dfNumeriquesSD.fillna(0)
    dfNumeriquesSDSansDataManquantNormalise = prep.StandardScaler().fit_transform(dfNumeriquesSDSansDataManquant)

    reduced_data = PCA(n_components=2).fit_transform(dfNumeriquesSDSansDataManquantNormalise)
    nbrs = NearestNeighbors(n_neighbors=6, algorithm=_algo).fit(reduced_data)
    distances, indices = nbrs.kneighbors(reduced_data)
    return distances, indices

def getMoySup(sup, distances):
    """
    Méthode qui calcule les moyennes de distance pour les "sup - 1" plus proches films à des fins de comparaison de méthode
    Arguments:
    sup -- valeur supérieure pour le range qui fait travailler les moyennes sur les sup - 1 films les plus proches pour chaqu
e film
    distances -- tableau des distances pour chaque film
    Retour:
    les distances moyennes pour chaque ième film le plus proche
    la moyenne des distances moyennes de chaque ième film le plus proche, une sorte de moyenne générale des distances pour le
s "sup - 1" films les plus proches
    """
    moyennes_distances = []
    for i in range(1,sup):
        moyennes_distances.append(np.mean(distances[:,i]))
    return moyennes_distances, np.mean(moyennes_distances)

def getMoyennesNouvellesVariables(_df):
    distances, _ = calculDistIndACPNN("kd_tree", _df)
    moyennes, moyenne = getMoySup(6, distances)
    print(moyennes)
    print("\nMoyenne générale des moyennes pour chaque ième film le plus proche : {0}".format(moyenne))

def comparatifNouvellesVariables():
    tab_moy = []
    tab_moy_moy = []
    tab_lib = []
    tab_moy.append([0.056189546983632661, 0.086320876090088552, 0.10810071891118363, 0.12424399225246316, 0.1393933494618962
4])
    tab_moy_moy.append(0.10284969674)
    tab_lib.append("Variables initiales")

    genres_num = dfOriginel['genres'].str.split('|').str.len()
    dfGenresNum = pd.DataFrame({'genre_num' : genres_num})
    dfOriginelGenresNum = pd.concat([dfOriginel, dfGenresNum], axis = 1)
    distances, _ = calculDistIndACPNN("kd_tree", dfOriginelGenresNum)
    moyennes, moyenne = getMoySup(6, distances)
    tab_moy.append(moyennes)
    tab_moy_moy.append(moyenne)
    tab_lib.append("Variables initiales + longueur genres")

    productif_num = dfOriginel['gross'] / dfOriginel['budget']
    dfProductifNum = pd.DataFrame({'productif_num' : productif_num})
    dfOriginelProductifNum = pd.concat([dfOriginel, dfProductifNum], axis = 1)
    distances, _ = calculDistIndACPNN("kd_tree", dfOriginelProductifNum)
    moyennes, moyenne = getMoySup(6, distances)
    tab_moy.append(moyennes)
    tab_moy_moy.append(moyenne)
    tab_lib.append("Variables initiales + ratio recette / budget")
```

```
    keywords_num = dfOriginel['plot_keywords'].str.split('|').str.len()
    dKeywordsNum = {'keywords_num' : keywords_num}
    dfKeywordsNum = pd.DataFrame(dKeywordsNum)
    dKeywordsNumAug = pd.concat([dfOriginel, dfKeywordsNum], axis = 1)
    distances, _ = calculDistIndACPNN("kd_tree", dKeywordsNumAug)
    moyennes, moyenne = getMoySup(6, distances)
    tab_moy.append(moyennes)
    tab_moy_moy.append(moyenne)
    tab_lib.append("Variables initiales + mots clefs")

    d = {'genre_num' : genres_num, 'productif_num' : productif_num, 'keywords_num' : keywords_num}
    df = pd.DataFrame(d)
    dfAug = pd.concat([dfOriginel, df], axis = 1)
    distances, _ = calculDistIndACPNN("kd_tree", dfAug)
    moyennes, moyenne = getMoySup(6, distances)
    tab_moy.append(moyennes)
    tab_moy_moy.append(moyenne)
    tab_lib.append("Variables initiales + ensemble longueur genre / ratio recette-budget / mots clefs")

    X = np.linspace(1,5,5)
    f, ax = plt.subplots(figsize=(8,8))

    text0 = TextArea(tab_lib[0], textprops=dict(color="red"))
    text1 = TextArea(tab_lib[1], textprops=dict(color="cyan"))
    text2 = TextArea(tab_lib[2], textprops=dict(color="blue"))
    text3 = TextArea(tab_lib[3], textprops=dict(color="green"))
    text4 = TextArea(tab_lib[4], textprops=dict(color="magenta"))
    text5 = TextArea("Moyenne des moyennes", textprops=dict(color="black"))
    box = VPacker(children=[text4, text3, text2, text1, text0, text5], pad=5, sep=5)
    anchored_box = AnchoredOffsetbox(loc=3, child=box, pad=0., bbox_to_anchor=(1.05, 0.73), bbox_transform=ax.transAxes,)
    ax.add_artist(anchored_box)
    plt.scatter(X, tab_moy[0], color='red')
    plt.scatter(X, tab_moy[1], color='cyan')
    plt.scatter(X, tab_moy[2], color='blue')
    plt.scatter(X, tab_moy[3], color='green')
    plt.scatter(X, tab_moy[4], color='magenta')
    plt.scatter(X, tab_moy_moy, color='black')
    ax.set_title(u'Distances moyennes pour chaque ième film')
    ax.set_xlabel(u'ième film le plus proche')
    ax.set_ylabel('distance')
    print(tab_moy_moy)

def comparatifAjoutVariables():
    tab_moy_moy = []
    for i in range(12, 29):
        distances, _ = calculDistIndACPNN("kd_tree", dfOriginel.iloc[:,0:i])
        _, moyenne = getMoySup(6, distances)
        tab_moy_moy.append(moyenne)

    X = np.linspace(12,28,17)
    f, ax = plt.subplots(figsize=(8,8))
    plt.scatter(X, tab_moy_moy, color='black')
    ax.set_title(u'Distances moyennes pour chaque groupe de variables')
    ax.set_xlabel(u'Nombre de variables utilisées dans le dataset')
    ax.set_ylabel('distance')

def grey_color_func(word, font_size, position, orientation, random_state=None, **kwargs):
    return "hsl(0, 0%%, %d%%)" % random.randint(60, 100)

def grapheWordCloud():
    genre_df = dfOriginel['genres'].str.split('|')
    liste_genres = []
    liste_genres_unique = []
    for element in genre_df:
        for genre in element:
            pos = genre.find("-")
            if pos > -1:
                genre = genre[:pos]+genre[pos+1:]
            liste_genres.append(genre)
            if genre not in liste_genres_unique:
                    liste_genres_unique.append(genre)

    stopwords = STOPWORDS.copy()

    nb_genre = len(liste_genres_unique)
    cv = CountVectorizer()
    cv_fit = cv.fit_transform(liste_genres)
    dic = cv.vocabulary_

    for genre in liste_genres_unique:
        val = dic[genre.lower()]
        del dic[genre.lower()]
        dic[genre] = nb_genre - 1 - val

    wc = WordCloud(max_words=2000, stopwords=stopwords, margin=10,random_state=1).generate_from_frequencies(dic)
    default_colors = wc.to_array()
    plt.title("Custom colors")
    plt.imshow(wc.recolor(color_func=grey_color_func, random_state=3), interpolation='bilinear')
    wc.to_file("wordcloud_figure.png")
    plt.axis("off")
    plt.figure()
    plt.title("Default colors")
    plt.imshow(default_colors)
    plt.axis("off")
    plt.show()
```

```
def getVoisinsProchesACP(_df, _method, _n_neighbors, _ind_film):
    reduced_data = PCA(n_components=2).fit_transform(_df)
    nbrs = NearestNeighbors(n_neighbors=_n_neighbors, algorithm=_method).fit(reduced_data)
    distances, indices = nbrs.kneighbors(reduced_data)
    return indices[_ind_film], distances[_ind_film]


def graphePositionProchesDansACP(_dfSDNorm, _dfSD, _ind_film):
    val_vect_propres = getPropre(_dfSDNorm, False)
    pp, _ = getVoisinsProchesACP(_dfSDNorm, "kd_tree", 6, _ind_film)
    print(pp)
    predict_score = np.where(
                        np.logical_or(
                            np.logical_or(
                                np.logical_or(
                                    np.logical_or(
                                        np.logical_or(_dfSD["movie_title"] == _dfSD.iloc[pp[0],11],
                                            _dfSD["movie_title"] == _dfSD.iloc[pp[1],11]),
                                        _dfSD["movie_title"] == _dfSD.iloc[pp[2],11]),
                                    _dfSD["movie_title"] == _dfSD.iloc[pp[3],11]),
                                _dfSD["movie_title"] == _dfSD.iloc[pp[4],11]),
                            _dfSD["movie_title"] == _dfSD.iloc[pp[5],11]), "Near", "Far")

    val_vect_propres.sort(key=lambda x: x[0], reverse=True)

    matrix_w = np.hstack((val_vect_propres[0][1].reshape(_dfSDNorm.shape[1],1), val_vect_propres[1][1].reshape(_dfSDNorm.shap
e[1],1)))

    Y = _dfSDNorm.dot(matrix_w)
    with plt.style.context('seaborn-whitegrid'):
        plt.clf()
        plt.figure(figsize=(6, 4))
        for lab, col in zip(('Far', 'Near'), ('oldlace', 'black')):
            plt.scatter(Y[predict_score==lab, 0], Y[predict_score==lab, 1], label=lab, c=col)
        plt.xlabel('Composante principale 1')
        plt.ylabel('Composante principale 2')
        plt.legend(loc='lower right')
        plt.show()

def getPosition(nom_var, pos_var, _ind_film):
    X = sorted(dfOriginelSansDoublon[nom_var], reverse=True)
    for i in range(len(X)):
        if X[i] == dfOriginelSansDoublon.iloc[_ind_film,pos_var]:
            break
    return i+1

def positions(_pp):
    print("num_voted_users")
    for i in _pp:
        print(getPosition('num_voted_users', 12, i))
    print("num_critic_for_reviews")
    for i in _pp:
        print(getPosition('num_critic_for_reviews', 2, i))
    print("cast_total_facebook_like")
    for i in _pp:
        print(getPosition('cast_total_facebook_likes', 13, i))
    print("director_facebook_likes")
    for i in _pp:
        print(getPosition('director_facebook_likes', 4, i))

def modeleAllVar(_df):
    dfNum = _df.select_dtypes(include = ['float64', 'int64']).iloc[:, :]
    dfNumSDSansDataManquant = dfNum.fillna(0)
    dfNumSDSansDataManquantNormalise = prep.StandardScaler().fit_transform(dfNumSDSansDataManquant)
    reduced_data = PCA(n_components=2).fit_transform(dfNumSDSansDataManquantNormalise)
    nbrs = NearestNeighbors(n_neighbors=6, algorithm="kd_tree").fit(reduced_data)
    _, indices = nbrs.kneighbors(reduced_data)
    ind = indices[0]
    ar = []
    for i in range(len(ind)):
        ligne = []
        ligne.append(_df.iloc[ind[i],11].replace(chr(0xa0), '').replace(chr(0xC2), ''))
        ligne.append(_df.iloc[ind[i],1])
        ligne.append(_df.iloc[ind[i],10])
        ligne.append(_df.iloc[ind[i],9])
        ar.append(ligne)
    dfTab = pd.DataFrame(ar, index = ind, columns = ['Film', 'réalisateur', 'acteur_1','genres'])
    print(dfTab.unstack())

def modeleVarGenres(_df):
    dfv =pd.DataFrame()
    genre_df = _df['genres'].str.split('|')
    liste_genres = []
    for element in genre_df:
        for genre in element:
            if genre not in liste_genres:
                liste_genres.append(genre)
    for genre in liste_genres:
        col_genre = np.where(_df['genres'].str.contains(genre) == True, 1, 0)
        label_genre = 'genre_'+genre
        d = {label_genre : col_genre}
        df_genre = pd.DataFrame(d)
        dfv = pd.concat([dfv, df_genre], axis = 1)
```

```
    dfNumSDSansDataManquant = dfv.fillna(0)
    dfNumSDSansDataManquantNormalise = prep.StandardScaler().fit_transform(dfNumSDSansDataManquant)
    reduced_data = PCA(n_components=2).fit_transform(dfNumSDSansDataManquantNormalise)
    nbrs = NearestNeighbors(n_neighbors=6, algorithm="kd_tree").fit(reduced_data)
    _, indices = nbrs.kneighbors(reduced_data)
    ind = indices[0]
    ar = []
    for i in range(len(ind)):
        ligne = []
        ligne.append(_df.iloc[ind[i],11].replace(chr(0xa0), '').replace(chr(0xC2), ''))
        ligne.append(_df.iloc[ind[i],1])
        ligne.append(_df.iloc[ind[i],10])
        ligne.append(_df.iloc[ind[i],9])
        ar.append(ligne)
    dfTab = pd.DataFrame(ar, index = ind, columns = ['Film', 'réalisateur', 'acteur_1','genres'])
    print(dfTab.unstack())

def modeleVarGenresReaActeur1(_df):

    dfv = pd.DataFrame()
    genre_df = _df['genres'].str.split('|')
    liste_genres = []
    for element in genre_df:
        for genre in element:
            if genre not in liste_genres:
                liste_genres.append(genre)
    for genre in liste_genres:
        col_genre = np.where(_df['genres'].str.contains(genre) == True, 1, 0)
        label_genre = 'genre_'+genre
        d = {label_genre : col_genre}
        df_genre = pd.DataFrame(d)
        dfv = pd.concat([dfv, df genre], axis = 1)
    dfc = _df[['director_name', 'actor_1_name']]
    cols = ['director_name', 'actor_1_name']
    dfAug5 = pd.get_dummies(dfc, columns = cols, prefix=['dn', 'an'])

    dfv2 = pd.concat([dfv, dfAug5], axis = 1, join_axes=[dfv.index])

    dfNumSDSansDataManquant = dfv2.fillna(0)
    dfNumSDSansDataManquantNormalise = prep.StandardScaler().fit_transform(dfNumSDSansDataManquant)
    reduced_data = PCA(n_components=2).fit_transform(dfNumSDSansDataManquantNormalise)
    nbrs = NearestNeighbors(n_neighbors=6, algorithm="kd_tree").fit(reduced_data)
    _, indices = nbrs.kneighbors(reduced_data)
    ind = indices[0]

    ar = []
    for i in range(len(ind)):
        ligne = []
        ligne.append(_df.iloc[ind[i],11].replace(chr(0xa0), '').replace(chr(0xC2), ''))
        ligne.append(_df.iloc[ind[i],1])
        ligne.append(_df.iloc[ind[i],10])
        ligne.append(_df.iloc[ind[i],9])
        ar.append(ligne)
    dfTab = pd.DataFrame(ar, index = ind, columns = ['Film', 'réalisateur', 'acteur_1','genres'])
    print(dfTab.unstack())
```