Model 2: Random forest keyword feature extraction and prediction

Data Prep

1. Merged the file of all keywords with the file containing genres using R.
2. Drop all irrelevent columns, keeping only group status (0/1) and keywords (0/1).

Random Forest

1. drop irrelevant
2. create test (80%) and train (20%)
3. create x and y variables. X is the keywords, y is the genre group (group 1 or not, group 2 or not...)
4. implement a random forest on each genre.
5. collect the indices of the top 20 more important keywords to predicting each genre, and print the variable name of these indices. This is the keyword
6. predict a random forest on the test set.
7. Collect the Haming Loss, % exact matches and % at-least-one match

R Code for Merging dataa<- read.csv("~/Downloads/all_movies_combine_with_keywords.csv") Monday<- read.csv("~/Downloads/full_movie_merge_genres.csv") new <- merge(x=dataa, y=Monday, by.x=("id"), by.y= ("tmdb_id"), all.y = TRUE)

In [34]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import hamming_loss
```

In [2]:

```
# import the dataset, using Pandas.
keywords_data = pd.read_csv("./new3.csv")
```

In [10]:

```
#### Keep only the keyword columns #########

# drop imdb features
imdb_cols = list(range(12030,12045))
data = keywords_data.drop(keywords_data.columns[imdb_cols],axis=1)

# drop tmdb features
tmdb_cols =  list(range(2,16))
data = data.drop(data.columns[tmdb_cols],axis=1)
data = data.drop(data.columns[0],axis=1)
```

In [11]:

```
data.head()
```

...

In [16]:

```
#### create training and testing sets
msk = np.random.rand(len(data)) < 0.8
train = data[msk]
test = data[~msk]
```

In [132]:

```
# select keyword columns
col_names= list(train1.columns.values)
#this is columns 16 to 12045
keyword_columns= col_names[2:-7]

#want cols Res to be genre type
colsRes1 = ['group1']
colsRes2 = ['group2']
colsRes3 = ['group3']
colsRes4 = ['group4']
colsRes5 = ['group5']
colsRes6 = ['group6']
colsRes7 = ['group7']
```

In [138]:

```
trainArr = train1.as_matrix(keyword_columns) #training array
trainRes1 = train1.as_matrix(colsRes1)
trainRes2 = train1.as_matrix(colsRes2)
trainRes3 = train1.as_matrix(colsRes3)
trainRes4 = train1.as_matrix(colsRes4)
trainRes5 = train1.as_matrix(colsRes5)
trainRes6 = train1.as_matrix(colsRes6)
trainRes7 = train1.as_matrix(colsRes7)
```

In [156]:

```
trainsets = [trainRes1, trainRes2, trainRes3, trainRes4, trainRes5, trainRes6, trai
```

...

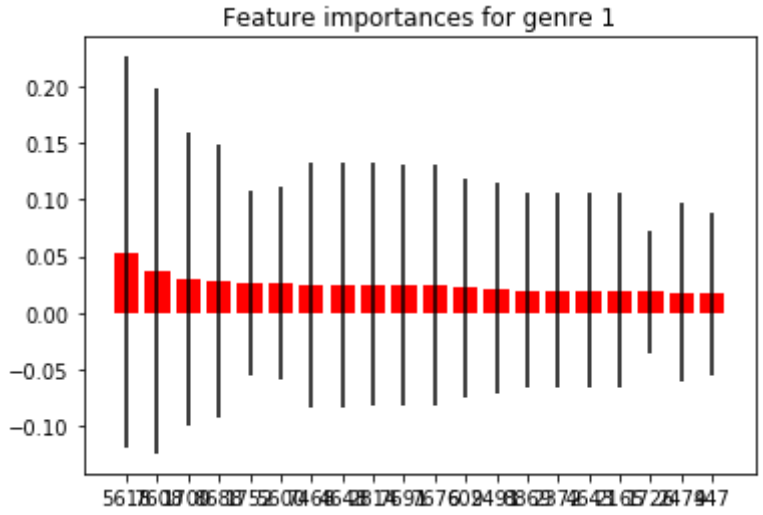
In [18]:

```
# select keyword columns
col_names= list(train.columns.values)
#this is columns 16 to 12045
keyword_columns= col_names[2:-7]

X_train = train.as_matrix(keyword_columns) #training array
X_test = test.as_matrix(keyword_columns) #training array
```

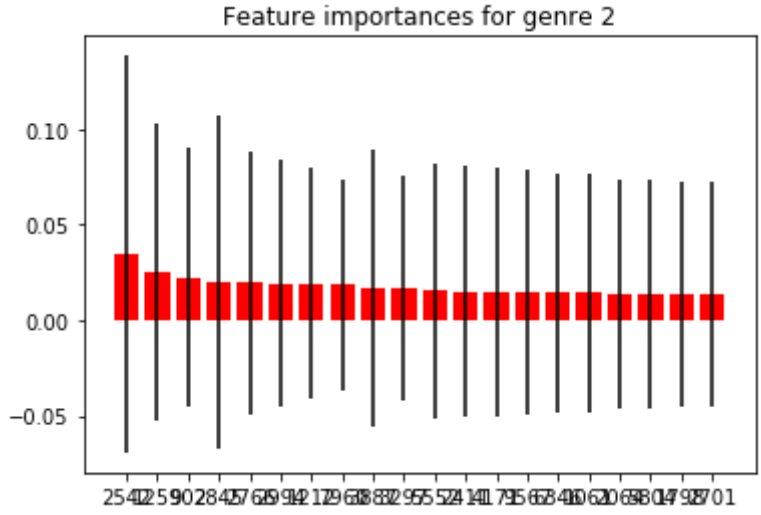
In [32]:

```
for i in range(1,8):

    print('\nGENRE ', i, "\n====================\n")
    Y_train = train['group' + str(i)].values

    rf = RandomForestClassifier(n_estimators=100)
    clf = RandomForestClassifier(n_estimators=20, max_depth=5)
    clf.fit(X_train, Y_train)

    # extract out feature importances
    importances = clf.feature_importances_
    std = np.std([tree.feature_importances_ for tree in clf.estimators_],
             axis=0)
    indices = np.argsort(importances)[::-1]

    plt.figure()
    # plot the feature importances of the forest
    plt.title("Feature importances for genre " + str(i))
    plt.bar(range(20), importances[indices[:20]],
       color="r", yerr=std[indices[:20]], align="center")
    plt.xticks(range(20), indices, )
    plt.show()

    # print the feature ranking
    print("Feature ranking:")
    for indx, i in enumerate(indices[:10]):
        print (indx+1, ": ", keyword_columns[i])
```

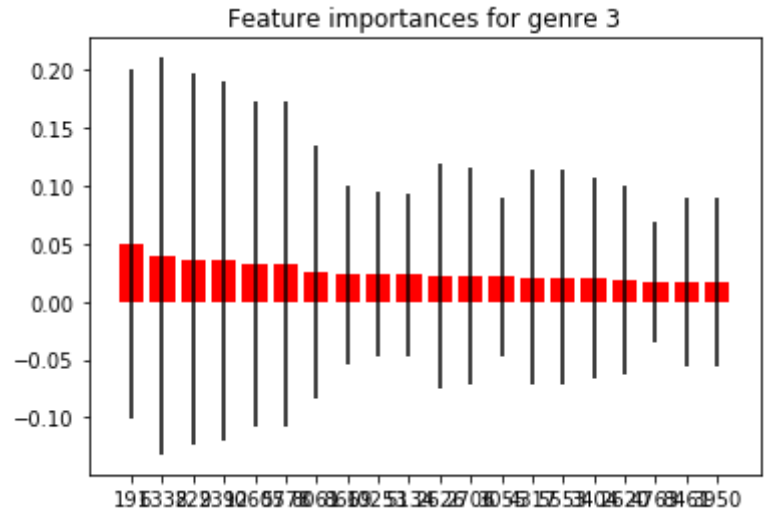
```
GENRE  1
====================
```



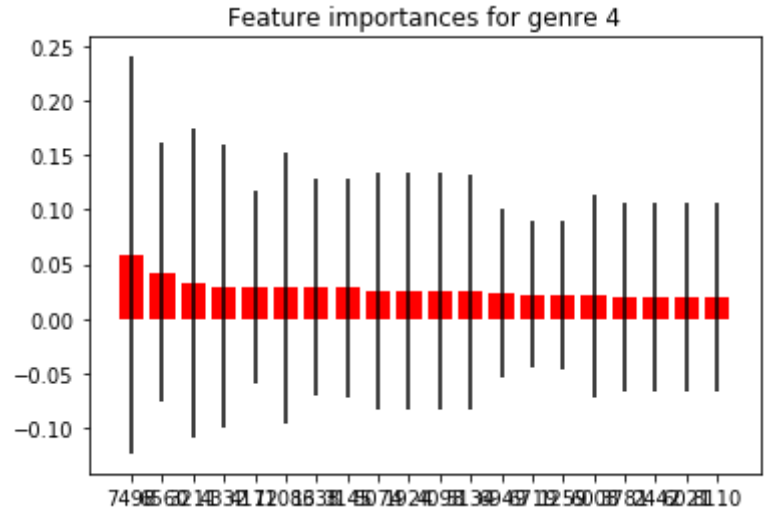
```
Feature ranking:
1 :  humanoid.robot
2 :  mountain.climbing
3 :  canine
4 :  pirate.ship
5 :  car.theft
6 :  human.animal.relationship
7 :  mods
8 :  gang.leader
9 :  cycle
10 :  muscle.car

GENRE  2
======================
```



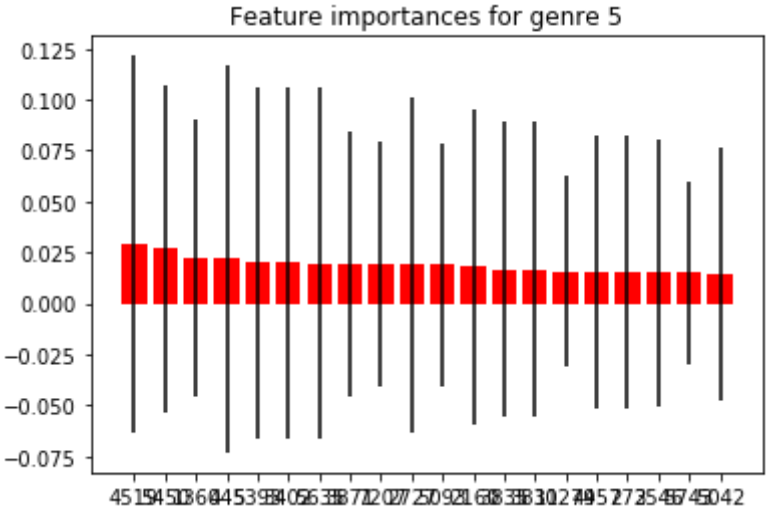

```
Feature ranking:
1 :  corpse.in.freezer
2 :  boat.accident
3 :  based.on.radio.show
4 :  dance.crew
5 :  cult.figure
6 :  defeat
7 :  blood.on.shirt
8 :  nightlife
9 :  expert
10 :  don.quixote

GENRE  3
======================
```



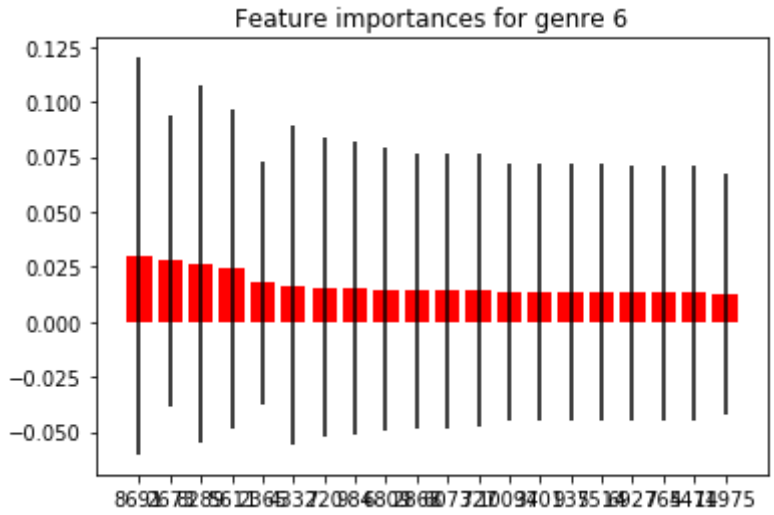

```
Feature ranking:
1 :  albatros
2 :  king.of.england
3 :  bank.manager
4 :  reality.tv
5 :  skopje
6 :  incommunicability
7 :  nursing.home
8 :  pinhead
9 :  separation
10 :  half.breed

GENRE  4
======================
```



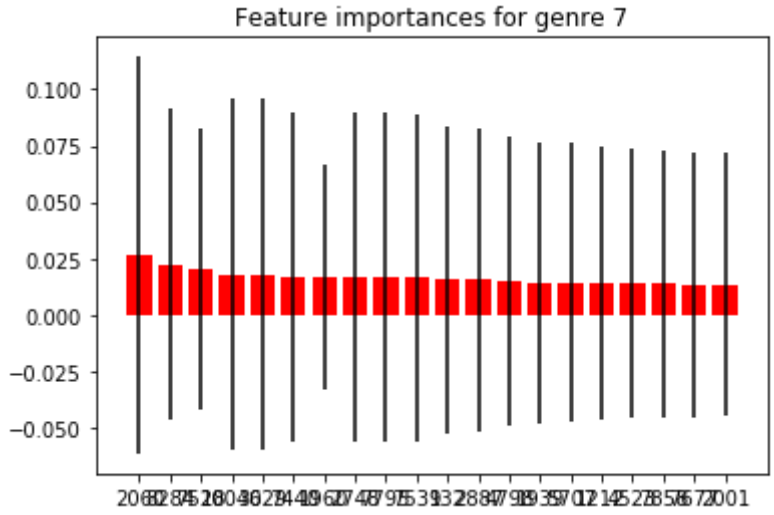
```
Feature ranking:
1 :  monkey.warrior
2 :  letter.opener
3 :  disobeying.orders
4 :  fjord
5 :  female.prime.minister
6 :  stephen.king
7 :  bosnian.war
8 :  dining.hall
9 :  guinevere
10 :  challenger

GENRE  5
======================
```




```
Feature ranking:
1 :  freedom.fighter
2 :  holocaust
3 :  bourbon
4 :  apache
5 :  hipster
6 :  drone.operators
7 :  hunting.trip
8 :  expectant.father
9 :  medium
10 :  cruise.liner.starship

GENRE  6
======================
```




```
Feature ranking:
1 :  pistol.duel
2 :  crime.novelist
3 :  overleven
4 :  human.rights
5 :  comic.book.collector
6 :  fjord
7 :  avante.garde.art
8 :  beauty.pageant
9 :  lottery
10 :  daredevil

GENRE  7
======================
```



```
Feature ranking:
1 :  child.s.point.of.view
2 :  overdose
3 :  mooning
4 :  scat
5 :  demo
6 :  mistress
7 :  charlatan
8 :  cthulhu
9 :  national.lampoon.serie
10 :  moonshine
```

In [35]:

```
X_test = test.as_matrix(keyword_columns)
Y_test = test.as_matrix(col_names[-7:])
Y_train = train.as_matrix(col_names[-7:])
```

In [41]:

```
## Prediction
rf = RandomForestClassifier(n_estimators=100)
y_test_pred = OneVsRestClassifier(rf).fit(X_train, Y_train).predict(X_test)
```

In [42]:

```
# These are how we measure error - Haming Loss, % exact matches and % at-least-one 
def error_measures(ypred, ytest):
    ypred = np.array(ypred)
    ytest = np.array(ytest)
    # Hamming loss
    from sklearn.metrics import hamming_loss
    h_loss = hamming_loss(ytest, ypred)

    # Percent exact matches
    y_pred_str = np.array([str(yi) for yi in ypred])
    y_test_str = np.array([str(yi) for yi in ytest])
    percent_exact = np.sum(y_pred_str == y_test_str) * 1. / ytest.shape[0]

    # Percent at least one match (at least one of the genres are both 1)
    atleastone_count = 0
    for ind in range(len(ypred)):
        yi_pred = ypred[ind]
        yi_test = ytest[ind]
        for i in range(len(yi_pred)):
            if yi_pred[i] == 1 and yi_test[i] == 1:
                atleastone_count += 1
                break
    percent_atleastone = atleastone_count * 1. / ytest.shape[0]

    return h_loss, percent_exact, percent_atleastone
```

In [44]:

```
error_measures(Y_test, y_test_pred)
```

Out[44]:

```
(0.19137466307277629, 0.27830188679245282, 0.7629716981132075)
```

The precentage exact match is much higher than with SVM although the percent at least one and hamming loss are lower

In [ ]:

```

```