# Model 1: SVM and RF on movie metadata

In Model 1, we use One vs Rest on the following columns:

- vote_count
- vote_average
- imdb_votes
- certificate (one-hot encoded and cleaned)
- num_stunts
- num_fx
- decade (one-hot encoded)
- month (one-hot encoded)

**We will use Radial SVM and Random Forest**

First we define our loss functions:

In [5]:

```
# These are how we measure error - Haming Loss, % exact matches and % at-least-one r
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
```

Next we prepare the data for our model:

In [6]:

```
import pandas as pd
import numpy as np

# Read in the data
train = pd.read_csv('../data/train_data_with_sampling.csv')
test = pd.read_csv('../data/test_data.csv')

# Split into train and test
X_train = train[['vote_count', 'vote_average', 'imdb_votes', 'certificate', 'num_stu
y_train = train[['group1', 'group2', 'group3', 'group4', 'group5', 'group6', 'group
X_test = test[['vote_count', 'vote_average', 'imdb_votes', 'certificate', 'num_stun
y_test = test[['group1', 'group2', 'group3', 'group4', 'group5', 'group6', 'group7'

# One hot encoding
X_train = pd.get_dummies(data=X_train,columns = ['certificate', 'decade', 'month'])
X_test = pd.get_dummies(data=X_test,columns = ['certificate', 'decade', 'month'])
```

## Radial SVM

In [104]:

```
# Use SVM to predict multi-class
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC

y_pred = OneVsRestClassifier(SVC(random_state=0)).fit(X_train, y_train).predict(X_te

hamming_loss, percent_exact, percent_atleastone = error_measures(y_pred, y_test)
print 'Untuned Radial SVM\n====================\n'
print 'Hamming loss: ', hamming_loss
print 'Percent exact: ', percent_exact
print 'Percent at least one: ', percent_atleastone
```

```
Untuned Radial SVM
====================

Hamming loss:  0.0856756756757
Percent exact:  0.621621621622
Percent at least one:  0.891621621622
```

In [48]:

```
# Tune SVM

from sklearn.model_selection import KFold
from sklearn.metrics import hamming_loss

kf = KFold(n_splits=4, random_state=428)

# Use Grid Search CV
C = [0.01, 1, 10, 100]
gamma = [0.000001, 0.001, 0.1, 1]

losses = []
for Ci in C:
    for gi in gamma:
        # Cross-validate
        losses_i = []
        for train_index, test_index in kf.split(X_train):
            X_train_, X_test_ = X_train.iloc[train_index], X_train.iloc[test_index]
            y_train_, y_test_ = y_train.iloc[train_index], y_train.iloc[test_index]
            y_pred_ = OneVsRestClassifier(SVC(random_state=0, C=Ci, gamma=gi)).fit(
                X_train_, y_train_).predict(X_test_)
            loss = hamming_loss(y_test_, y_pred_)
            losses_i.append(loss)
        losses.append({'C': Ci, 'gamma': gi, 'cv_score': np.mean(losses_i)})
```

In [55]:

```
# Check the losses to choose the best C and gamma for Radial SVM
print 'Best performing C and gamma after tuning\n=====================\n'
best_dict = sorted(losses, key=lambda k: k['cv_score'])[0]
print best_dict
chosen_C = best_dict['C']
chosen_gamma = best_dict['gamma']
```

```
Best performing C and gamma after tuning
=====================

{'C': 10, 'cv_score': 0.12451346030757754, 'gamma': 0.1}
```

In [105]:

```
# Re-run SVM on tuned hyperparams

y_pred = OneVsRestClassifier(SVC(random_state=0, C=chosen_C, gamma=chosen_gamma)).f

hamming_loss, percent_exact, percent_atleastone = error_measures(y_pred, y_test)
print 'Tuned Radial SVM with C=%f and gamma=%f\n====================\n' % (chosen_
print 'Hamming loss: ', hamming_loss
print 'Percent exact: ', percent_exact
print 'Percent at least one: ', percent_atleastone
```

```
Tuned Radial SVM with C=10.000000 and gamma=0.100000
=====================

Hamming loss:  0.0824324324324
Percent exact:  0.630540540541
Percent at least one:  0.888108108108
```

After tuning our Radial SVM, percent exact went up to 63% and hamming loss went down slightly! However, it seems the percent of at least one genre matching dipped slightly.

## Random Forest

In [108]:

```
# Use RandomForest to predict multi-class

from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=25, random_state=0)
y_pred = OneVsRestClassifier(clf).fit(X_train, y_train).predict(X_test)

hamming_loss, percent_exact, percent_atleastone = error_measures(y_pred, y_test)
print 'Untuned Random Forest\n====================\n'
print 'Hamming loss: ', hamming_loss
print 'Percent exact: ', percent_exact
print 'Percent at least one: ', percent_atleastone
```

```
Untuned Random Forest
=====================

Hamming loss:  0.0935135135135
Percent exact:  0.606486486486
Percent at least one:  0.947297297297
```

In [69]:

```
# Tune Random Forest

from sklearn.model_selection import KFold
from sklearn.metrics import hamming_loss

kf = KFold(n_splits=4, random_state=428)

# First we test the number of trees necessary

num_trees = np.arange(5,100,5)
tree_losses = []

for num_tree in num_trees:
    # Cross-validate
    losses_i = []
    for train_index, test_index in kf.split(X_train):
        X_train_, X_test_ = X_train.iloc[train_index], X_train.iloc[test_index]
        y_train_, y_test_ = y_train.iloc[train_index], y_train.iloc[test_index]
        clf = RandomForestClassifier(n_estimators=num_tree, n_jobs=-1, random_state=
        y_pred_ = OneVsRestClassifier(clf).fit(X_train_, y_train_).predict(X_test_)
        loss = hamming_loss(y_test_, y_pred_)
        losses_i.append(loss)
    tree_losses.append(np.mean(losses_i))
```

In [81]:

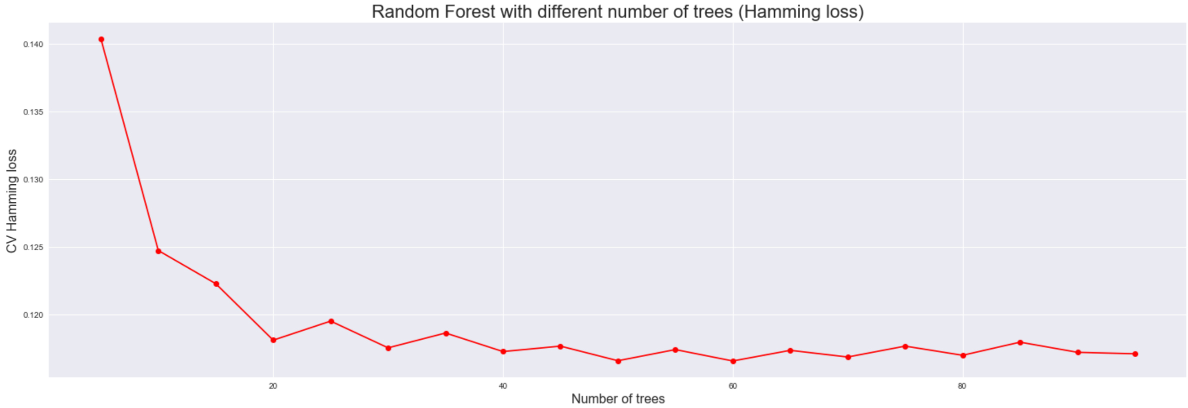
```
# We plot tree_losses, and find the lowest number of trees necessary

import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

# plt.plot(tree_losses)
plt.figure(figsize=(25,8))
plt.title('Random Forest with different number of trees (Hamming loss)', fontsize=2
plt.plot(num_trees, tree_losses, 'ro-')
plt.xlabel('Number of trees', fontsize=16)
plt.ylabel('CV Hamming loss', fontsize=16)
```

Out[81]:

```
<matplotlib.text.Text at 0x119668950>
```



It seems like 25 trees is more than sufficient.

In [91]:

```
# Tune the rest

from sklearn.model_selection import KFold
from sklearn.metrics import hamming_loss

kf = KFold(n_splits=4, random_state=428)

num_tree_optimal = 25
max_features = ["sqrt", 0.2, 0.5, "log2", "auto"]
min_sample_leaves = [1,2,5,10,20]

losses_rf = []

for max_feature in max_features:
    for min_sample_leaf in min_sample_leaves:
        # Cross-validate
        losses_i = []
        for train_index, test_index in kf.split(X_train):
            X_train_, X_test_ = X_train.iloc[train_index], X_train.iloc[test_index]
            y_train_, y_test_ = y_train.iloc[train_index], y_train.iloc[test_index]
            clf = RandomForestClassifier(n_estimators=num_tree_optimal, random_stat
                                         min_samples_leaf=min_sample_leaf, max_featu
            y_pred_ = OneVsRestClassifier(clf).fit(X_train_, y_train_).predict(X_te
            loss = hamming_loss(y_test_, y_pred_)
            losses_i.append(loss)
        losses_rf.append({'Max_feature': max_feature, 'Min_sample_leaf': min_sample_
                          'Loss': np.mean(losses_i)})
```

In [92]:

```
# Check the losses to choose the best max_features and min_samples_leaf for RF
print 'Best performing RF hyperparams after tuning\n=====================\n'
best_dict = sorted(losses_rf, key=lambda k: k['Loss'])[0]
print best_dict
max_feature_optimal = best_dict['Max_feature']
min_sample_optimal = best_dict['Min_sample_leaf']
```

```
Best performing RF hyperparams after tuning
=====================

{'Loss': 0.11952204859227675, 'Min_sample_leaf': 1, 'Max_feature': 'sq
rt'}
```

In [107]:

```
# Re-run RF on tuned hyperparams

from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=200, random_state=0, min_samples_leaf=min_
                             max_features=max_feature_optimal)
y_pred = OneVsRestClassifier(clf).fit(X_train, y_train).predict(X_test)

hamming_loss, percent_exact, percent_atleastone = error_measures(y_pred, y_test)
print 'Tuned Random Forest\n=====================\n'
print 'Hamming loss: ', hamming_loss
print 'Percent exact: ', percent_exact
print 'Percent at least one: ', percent_atleastone
```

```
Tuned Random Forest
=====================

Hamming loss:  0.0888803088803
Percent exact:  0.615135135135
Percent at least one:  0.953783783784
```

After tuning, all error measures improved. **It is interesting to note that SVM is better at predicting exact matches (higher percent exact), while RF is superior in predicting percent at least one.**

## Artificial Neural Networks

In [7]:

```
# Let's see if we can use deep learning (ANN) for multi-label to bring error down

# Scale features first - This is particularly important for ANN
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_scaled = sc.fit_transform(X_train)
X_test_scaled = sc.transform(X_test)
y_train = np.array(y_train)
y_test = np.array(y_test)

# Hand tuned the following parameters below

# Build ANN
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout

# Initialize ANN
classifier = Sequential()

# Input layer and first hidden layer
n_in = X_train_scaled.shape[1]
n_out = y_train.shape[1] # Multi-label
n_hidden = (n_in + n_out)/2 # In + Out / 2 is a bit of an art

# Just one hidden layer is enough due to pretty simple dataset
classifier.add(Dense(units=n_hidden, kernel_initializer='uniform', activation='relu

# Output layer
classifier.add(Dense(units=n_out, kernel_initializer='uniform', activation='sigmoid

# Compile and fit ANN: Adam SGD, Log loss because of sigmoid function
classifier.compile(optimizer='adam', loss='categorical_crossentropy')
classifier.fit(X_train_scaled, y_train, batch_size=5, epochs=50) # Very small batch
```

```
Using TensorFlow backend.

Epoch 1/50
7814/7814 [==============================] - 3s - loss: 4.7670
Epoch 2/50
7814/7814 [==============================] - 3s - loss: 4.6038
Epoch 3/50
7814/7814 [==============================] - 3s - loss: 4.5630
Epoch 4/50
7814/7814 [==============================] - 4s - loss: 4.5321
Epoch 5/50
7814/7814 [==============================] - 3s - loss: 4.5080
Epoch 6/50
7814/7814 [==============================] - 3s - loss: 4.4868
Epoch 7/50
7814/7814 [==============================] - 3s - loss: 4.4697
Epoch 8/50
7814/7814 [==============================] - 3s - loss: 4.4536
Epoch 9/50
7814/7814 [==============================] - 3s - loss: 4.4406
Epoch 10/50
```

```
Epoch 10/50
7814/7814 [==============================] - 3s - loss: 4.4305
Epoch 11/50
7814/7814 [==============================] - 3s - loss: 4.4170
Epoch 12/50
7814/7814 [==============================] - 3s - loss: 4.4124
Epoch 13/50
7814/7814 [==============================] - 3s - loss: 4.4053
Epoch 14/50
7814/7814 [==============================] - 3s - loss: 4.3978
Epoch 15/50
7814/7814 [==============================] - 3s - loss: 4.3926
Epoch 16/50
7814/7814 [==============================] - 3s - loss: 4.3864
Epoch 17/50
7814/7814 [==============================] - 3s - loss: 4.3822
Epoch 18/50
7814/7814 [==============================] - 3s - loss: 4.3762
Epoch 19/50
7814/7814 [==============================] - 3s - loss: 4.3727
Epoch 20/50
7814/7814 [==============================] - 3s - loss: 4.3683
Epoch 21/50
7814/7814 [==============================] - 3s - loss: 4.3645
Epoch 22/50
7814/7814 [==============================] - 3s - loss: 4.3592
Epoch 23/50
7814/7814 [==============================] - 3s - loss: 4.3572
Epoch 24/50
7814/7814 [==============================] - 3s - loss: 4.3525
Epoch 25/50
7814/7814 [==============================] - 3s - loss: 4.3490
Epoch 26/50
7814/7814 [==============================] - 3s - loss: 4.3449
Epoch 27/50
7814/7814 [==============================] - 3s - loss: 4.3427
Epoch 28/50
7814/7814 [==============================] - 3s - loss: 4.3411
Epoch 29/50
7814/7814 [==============================] - 4s - loss: 4.3390
Epoch 30/50
7814/7814 [==============================] - 3s - loss: 4.3362
Epoch 31/50
7814/7814 [==============================] - 3s - loss: 4.3330
Epoch 32/50
7814/7814 [==============================] - 3s - loss: 4.3321
Epoch 33/50
7814/7814 [==============================] - 3s - loss: 4.3275
Epoch 34/50
7814/7814 [==============================] - 3s - loss: 4.3258
Epoch 35/50
7814/7814 [==============================] - 4s - loss: 4.3262
Epoch 36/50
7814/7814 [==============================] - 3s - loss: 4.3231
Epoch 37/50
7814/7814 [==============================] - 4s - loss: 4.3221
Epoch 38/50
7814/7814 [==============================] - 4s - loss: 4.3206
Epoch 39/50
7814/7814 [==============================] - 3s - loss: 4.3167
```