PA03: Ensemble Learning (Due Nov 16)

In this programming assignment you will implement and experiment with learning ensembles in recommender systems.

The MovieLens Dataset

We are using the movielens dataset obtained here http://www.grouplens.org/node/73. The dataset consists of 100k ratings by ~900 users on ~1500 items. The idea is to predict user ratings on a given item, so each (user,item) pair is an example, features include user and item features.

We provide a training dataset of ~90,000 movie recommendations for your use in this assignment. You can look at file `pa03/util/prep_data.py` to see the code that was used to create the dataset. The datatable is a pickled pandas `DataFrame` located at `movie_data/ratings_train.pda`. We also provide a test dataset of ~9500 ratings in file `movie_data/ratings_test.pda`.

In [1]:
```
# this is useful when updating code outside the notebook
#%load_ext autoreload
#%autoreload 2
```

In [2]:
```
# This was run to prepare the data, don't run unless you have downloaded the data
# from url above

#from pa03.util.prep_data import prep_data
#ratings = prep_data('../devel/ml-100k', 'ua.base', None)
#ratings.pop('videodate')
#ratings.pop('unknown')
#print ratings.shape
#ratings.save('movie_data/ratings_train.pda')
```

In [3]:
```
#ratings = prep_data('../devel/ml-100k', 'ua.test', None)
#ratings.pop('videodate')
#ratings.pop('unknown')
#print ratings.shape
#ratings.save('movie_data/ratings_test.pda')
```

Let's load the data and get do some exploration

In [4]:
```
import pandas as pd
ratings=pd.load('movie_data/ratings_train.pda')

nratings = ratings.shape[0]
print 'there are %d ratings' % nratings

# the variables
print ratings.columns
```

```
there are 90570 ratings
array([userid, itemid, rating, age, gender, occupation, Action, Adventure,
       Animation, Children's, Comedy, Crime, Documentary, Drama, Fantasy,
       Film-Noir, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War,
       Western, decade, isgood], dtype=object)
```

In [5]:
```
# you can get more info from prep_data itself
```

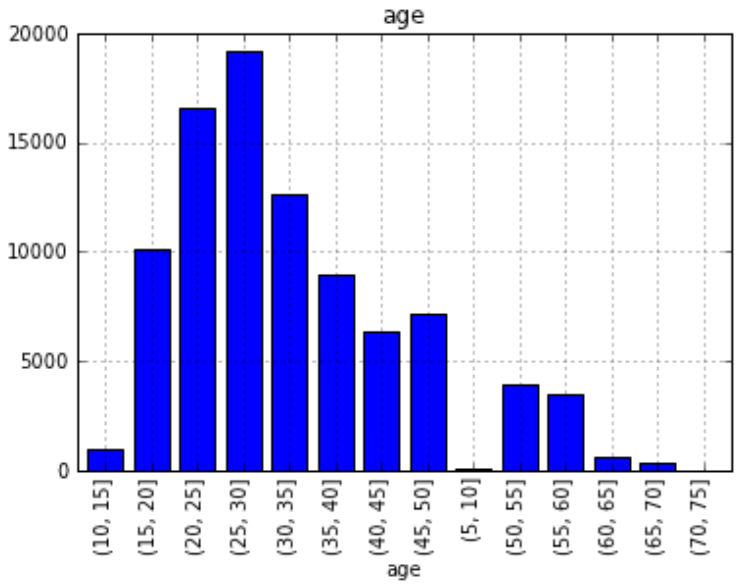
```
from pa03.util.prep_data import prep_data
?prep_data
```

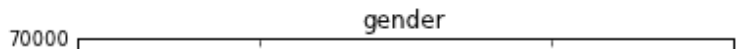
The columns are fairly self-explanatory:

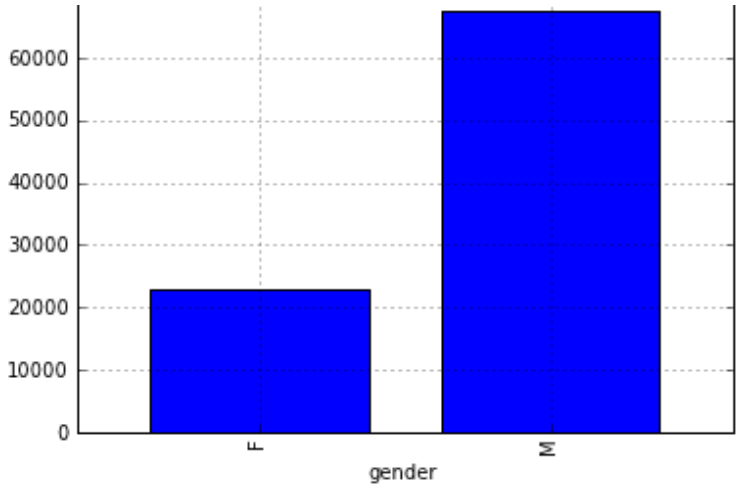
- rating: a rating from 1-5
- age: user's age (numeric)
- gender: user's gender (string, "M" or "F")
- occupation: user's occupation (string)
- Action,...,Western: binary (0,1) variables indicating movie genre
- decade: decade in which movies were released (numeric e.g., 90)
- isgood: binary (+1,-1) variable indicating if rating > 3

Question 1: Fill in the code below to get some information about the dataset

In [6]:
```
# example: get a histogram of the number of ratings per age group in the dataset
figure()
ratings.groupby('age').size().plot(kind='bar')
title('age')
show()

# example: how many ratings are there per gender?
figure()
ratings.groupby('gender').size().plot(kind='bar')
title('gender')
show()

# COMPLETE: how many ratings are there per occupation?

# COMPLETE: how many ratings are there per decade?

# COMPLETE: what is the number of good movies and bad movies (isgood==+1, isgood==-1

# COMPLETE: how many examples per rating value (1-5) are there?

# COMPLETE: how many ratings per genre are there? (hint look at the pandas.DataFrame
```







In [7]:
```
# COMPLETE: how many ratings are there for dramas from the 90's from 25-30 year old

# COMPLETE: what is the proportion of good movies (isgood == +1) for these ratings
```

Collaborative Filtering

Modern recommender systems use both information about user's and items (content-based system) along with previous ratings. The challenge is how to incorporate previous ratings into the recommender system. As an example, suppose you want to predict if user `Joe` will rate movie `Weekend at Bernie's` as a good movie. One way is to see how all other users have rated `Weekend at Bernie's` and use those rankings in your classifier (for example, by using the movie's mean rating).

A slightly more sophisticated way of doing it is to weight each user's rating by how similarly they rate movies to `Joe`. To implement this we need to define user similarities. One way of doing this is to represent each user by the vector of ratings they have made on movies and use the cosine between these vectors as the similarity. For instance, if the ratings for user `Joe` is denoted as x and the ratings for user `Sandy` is denoted as y, then the similarity between `Joe` and `Sandy` is

$$w_{x,y} = \frac{x \cdot y}{\|x\|_2 \|y\|_2}$$

To determine `Joe's` rating for the movie, we find the most similar users to `Joe`, say 20 of them, and calculate the average rating for `Weekend at Bernies` for these users, weighted by each user's similarity to `Joe`.

We have provided code to calculate user similarities and provide similarity weighted ratings.

In [8]:
```
from pa03.util.cfiltering import CFilter

# get info about CFilter
?CFilter

# and the rating function
?CFilter.get_cf_rating
```

In [9]:
```
# create object on the first 10,000 ratings (you may want to build this on the compl
# of ratings to answer the question below)
cf=CFilter(ratings.ix[:10000,:])
```

In [10]:
```
# get predicted ratings
cf_ratings=cf.get_cf_rating(ratings)
```

```
# print the first 10
print cf_ratings[:10]
```

```
[ 2.88679245  4.          4.125       4.16666667  3.42105263  4.5
  4.10526316  4.35714286  3.66666667  3.33333333]
```

Question 2: How good are these predicted ratings? How well can you predict if you get a good rating using this? (I am asking about training set accuracy here, you'll answer questions about generalization error later).

Question 3: Extend the code for collaborative filtering in file `pa03.util.cfiltering.py` to implement item-item similarity (e.g., compute `Joe's` rating for `Weekend at Bernie's` from `Joe's` ratings for similar movies, again using cosine similarity).

Question 4: Are predicted ratings from item-item similarity better?

Hybrid recommender systems

We can combine predicted ratings from either (or both) user-user and item-item similarities with user and item features and construct rating predictors. Using the code provided we would prepare a dataset for this prediction task as follows:

In [11]:
```
# remove the userid, itemid and rating column from the training set
ratings.pop('userid')
ratings.pop('itemid')
_dont_print = ratings.pop('rating')
```

In [12]:
```
# add discretized predicted ratings from CFilter
# discretization is from 0-5 in .25 steps
ratings['cf_rating']=pd.cut(cf_ratings,np.arange(-1,6,.25))

# now ratings is ready for prediction
```

A slight point, you should consider building the CFilter object part of training so, to construct a test set you would do something like this:

In [13]:
```
test_ratings=pd.load('movie_data/ratings_test.pda')

# add ratings to the test set using the cf object built on the training set
test_ratings['cf_rating']=pd.cut(cf.get_cf_rating(test_ratings), np.arange(-1,6,.25))

# remove columns not used in 'isgood' prediction
test_ratings.pop('userid')
test_ratings.pop('itemid')
_dont_print=test_ratings.pop('rating')

# test_ratings is ready for prediction
```

Question 5: Compare the following versions of your recommender system:

I. content-based rec system (using only user and item features)
II. using only predicted ratings from item-item similarities
III. using only predicted ratings from user-user similarities
IV. using only predicted ratings from both item-item and user-user similarities
V. content-based + item-item predicted ratings
VI. content-based + user-user predicted ratings
VII. content-based + item-item + user-user predicted ratings

Note 1: You need to choose a classifier to run this test. We have provided a decision tree learner in module

`pa03.dectree.DTree`. It only handles categorical data (recall all features in this dataset are categorical), and uses the Gini index for scoring splits. You can use this or your decision tree code from PA01.

Note 2: Implement the bootstraped F-score metric from Algorithm 20 (pg. 65) of the book and use it to perform this comparison (i.e., provide confidence intervals).

Note 3: You should make the bootstrap sampling code usable beyond this question as you will need it again when implementing Bagging and Random Forest ensembles.

In [14]:
```
# get more info about decision tree learning
from pa03.dectree.DTree import DTree
?DTree
```

In [15]:
```
# this function is a handy wrapper to use
from pa03.dectree.DTree import get_tree
?get_tree
```

Ensemble Learning

Now you will implement three ensemble learning methods to see if you can improve on results from the previous section. You should stick to version VII of the recommender system (content-based + item-item + user-user predicted ratings) from this point forward. Continue using your bootstraped F-score to perform comparisons below. Provide a sketch discussion (with code snippets if needed) below indicating how you implemented each method.

Question 6: Implement Bagging where you bag decision trees. You should try to reuse your bootrsap code from the previous section. Compare bagged trees to your results from Question 5. Comment on how the number of trees in the ensemble affects prediction performance.

Question 7: Implement Random Forest learning. This requires that you can construct "random" trees where splits are done over randomly chosen subset of candidate splits. Feel free to extend the decision tree code we provide (look at `pa03.dectree.scoring.py`) to do this. Otherwise, modify your PA01 decision tree code. You should be able to reuse your bagging code from Question 6 here. Compare random forests to the Bagging (Question 6) and single Decision Tree (Question 5).

Question 8: Implement AdaBoost for decision tree stumps (trees with a single split). You need to be able to train weighted decision tree stumps to do this. Feel free to extend the decision tree code we provide, or use your PA01 decision tree code. You should be able to define weighted versions of the Gini index (if you use our code), or information gain (if you use your code) by changing how the proportion of examples from each class are calculated at a given split. Compare boosted stumps to all previous methods.

Handing in

As before, you should run all heavy computations outside of the IPython notebook. However, all code you run to include in your discussion (e.g., to make plots, compute bootstrapped f-score from saved predictions) can be run inside the notebook. Please submit your source code along with this notebook.