Plan for today
- General discussion of your proposals
- Sample project overview (what you have to turn in on Tuesday)
- More tools you might want to use
- More examples of past projects

General feedback on proposals
- We need more specifics on what exactly you’re planning to build.
  - Vagueness was fine for the proposals, but it’s not appropriate for your overview.
  - Avoid discussion of “possible applications” – your overview is a commitment to develop a fleshed-out, polished application.
  - Be ambitious but realistic. It’s okay if at some future point you realize that you don’t have time to implement every feature described in your overview, but your final product should not deviate too far from the scope of your overview.

General feedback on proposals
- Measurement criteria are essential
  - Creating a cool application is great but not sufficient – you also need a predetermined standard for evaluating the success or failure of your work.
  - Some kind of scientific numerical analysis of your system’s performance in comparison to a baseline or rival system:
    - precision/recall
    - user satisfaction ratings
    - correlation or mean squared error (if you’re predicting values)
    - processing time, main memory requirements, disk space

General feedback on proposals
- Remember: a successful project doesn’t have to achieve great performance!
  - Of course it’s better to get good results…
  - But there can be significant value in trying something interesting and finding that it doesn’t work very well.
  - So don’t be afraid to explore an idea that isn’t guaranteed to pan out – as long as there’s reason to believe that it might.

Project overview: Suggested structure
- Title
- Group members
- Abstract (one short paragraph)
- Topic(s) investigated
- Relevant prior work (paper citations, actual systems)
- Delineation of group member responsibilities
- Data sources
- Technologies (programming languages, software, etc.)
- Existing tools leveraged
- Implementation details
- Submission calendar:
  - Block 1
  - Block 2
  - Block 3 (final product)
Sample project overview
(idealized – not my actual proposal!)

- MovieThing: A web-based collaborative filtering movie recommendation system
- Group: Louis Eisenberg (CS coterm) and Joe User (CS senior)
- Abstract: I will conduct an online experiment by building a website on which registered users can provide ratings for popular movies using a graphical interface. Once I have collected ratings from a substantial number of users, I will generate movie recommendations, assigning each user randomly to one of a handful of distinct recommendation algorithms. I will then solicit feedback from the users on the quality of the recommendations and use that feedback to perform a qualitative analysis of the relative accuracy of the different algorithms.

Sample project overview

- Topics investigated: collaborative filtering, recommendation systems
- Relevant prior work:
  - MovieLens (U. of Minn.)
  - Jester (UC-Berkeley)
- CF research papers:
  - http://jamesthornton.com/cf/
  - More research papers…

Sample project overview

- Group member responsibilities:
  - Louis: set up database, JDBC and utility code, JavaScript sliders, evaluation code
  - Joe: AWS code, JSP and servlet front-end code, literature review
  - Both: fill movie table, design CF algorithms, recruit subjects, write final paper
- Data sources:
  - Movie data (title, actors, genres, etc.) from IMDB and Amazon
  - Movie ratings supplied by my users
  - Amazon product similarity data
- Technologies: servlets/JSP, Javascript, MySQL
- Existing tools leveraged: Amazon Web Services

Implementation details:

- Website will display movies in tabular format with ability to search/filter by title, genre, actors, etc. Users rate movies by dragging sliders.
- Algorithms:
  - Amazon: use product similarity to generate predicted ratings based on weighted averages using user’s ratings and movies considered “similar” to those the user has rated
  - Standard: predicted ratings are weighted averages using user’s Pearson correlation to other users and the ratings of the other users
  - General deviation: emphasize movies for which user has an unusual opinion by introducing additional term into covariance calculation (which factors into user similarity weight)
  - Personal deviation: emphasize movies about which user feels strongly by cubing covariance terms.
  - Both deviations: combine tweaks of general and personal.
- Evaluation:
  - Overall ratings of quality of recommendation lists
  - Correlation between predicted and actual ratings for recommended movies that user has already seen

Sample project overview

- Submission calendar:
  - Block 1:
    - movies table is fully populated
    - website is live and accepting ratings
  - Block 2:
    - sufficient users and ratings have been collected
    - Amazon similarity data has been retrieved
    - recommendation algorithms are functional
  - Block 3:
    - users have received recommendations and provided feedback
    - final paper includes analysis of algorithms’ relative performance

Notes on sample project overview

- Your overview should be more extensive than this sample…
  - More specific implementation details, particularly in regard to algorithms
  - More specific goals for each block/milestone
  - Contingency plans for slight modifications to your project if you encounter obstacles?
More tools

MALLET
- A Machine Learning for Language Toolkit
- http://mallet.cs.umass.edu/
- "an integrated collection of Java code useful for statistical natural language processing, document classification, clustering, information extraction, and other machine learning applications to text"
- Minimally documented but has lots of stuff:
  - Building feature vectors
  - Various classification methods (Naïve Bayes, max-ent, boosting, winnowing)
  - Evaluation: precision, recall, F1, etc.
  - N-grams
  - Selecting features using information gain
- They have some examples of front-end code

MinorThird
- http://minorthird.sourceforge.net/
- "a collection of Java classes for storing text, annotating text, and learning to extract entities and categorize text"
- Documentation seems to be pretty good: comprehensive Javadocs, tutorial, FAQ...
- Has the concept of "spans" (sequences of words) that can be extracted and classified based on content or context
- Stored documents can be annotated in independent files using TextLabels (denoting, say, part-of-speech and semantic information)

Weka 3: Data Mining Software in Java
- http://www.cs.waikato.ac.nz/ml/weka/
- "Weka is a collection of machine learning algorithms for data mining tasks. The algorithms can either be applied directly to a dataset or called from your own Java code. Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. It is also well-suited for developing new machine learning schemes."
- Has a GUI
- Extensive documentation
- Website lists a number of compatible datasets (regression and classification problems)
- Also lists many Weka-related projects

CLUTO
- http://www-users.cs.umn.edu/~karypis/cluto/
- "a software package for clustering low- and high-dimensional datasets and for analyzing the characteristics of the various clusters"
- Partitional, agglomerative and graph-partitioning algorithms
- Various similarity/distance metrics
- Many options/tools for visualizing and summarizing clustering results
- Claims to scale to hundreds of thousands of objects in tens of thousands of dimensions
- wCluto: web-based application built on CLUTO
- gCluto: cross-platform graphical application

MG4J: Managing Gigabytes for Java
- http://mg4j.dsi.unimi.it/
- "a collaborative effort aimed at providing a free Java implementation of inverted-index compression techniques; as a by-product, it offers several general-purpose optimised classes, including fast & compact mutable strings, bit-level I/O, fast unsynchronised buffered streams, (possibly signed) minimal perfect hashing for very large strings collections, etc."
Crawlers

- UbiCrawler
  - [http://ubi.imc.pi.cnr.it/projects/ubicrawler/](http://ubi.imc.pi.cnr.it/projects/ubicrawler/)
  - Not available publicly, but "upon agreement with the authors for scientific purposes."
  - Primary advantage: "a very effective assignment function (based on consistent hashing) for partitioning the domain to crawl"
- Teg Grenager’s crawler
  - See the links on the projects page of the course website
  - Easily extensible

TiMBL

- Tilburg Memory Based Learner
  - [http://ilk.kub.nl/software.html](http://ilk.kub.nl/software.html)
  - Nearest-neighbor classification software with lots of options:
    - k
    - voting scheme
    - feature weighting
    - optimizations
    - built-in leave-one-out testing and cross-fold validation

Stanford WebBase (more info)

- Kayur Patel will supply a Java client to the WebBase data. It should be available by next Tuesday
- WebBase provides the source for a client written in C

More links than you can shake a stick at

- Many options for all kinds of different NLP tools and tasks:
  - POS taggers
  - Probabilistic parsers
  - Named entity recognition
  - NP chunking
  - Information extraction/wrapper induction
  - Word sense disambiguation
  - Lots of datasets/corpora

Reminder: pubcrawl

- SULinux server
- Terabytes of disk space
- MySQL
- Tomcat upon request
- Email us if you want access

Tutorial on basic skills/tools

- Provides basic instructions for using Java and some of its key packages, Ant, CVS, MySQL, Lucene, Tomcat, etc.
- Mostly stuff that the majority of you already know, but definitely worth browsing through
More datasets

- Another place to look for data:
  `/usr/class/cs276a/data1`
  - `.../dmoz`
  - `.../selected-linguistic-data`
  - `.../linguistic-data`
    - This is a superset of the selected-linguistic-data directory, but you need permission to access it (we’ll take care of this soon)
    - More information on the contents of this directory at

Some more examples of projects from two years ago

Returning Multiple Pages as Individual Search Results

- Angrish and Malhotra
  - Idea: Find a group of logically linked documents that collectively satisfy the user’s information need
  - Logical link could be any number of things. They defined two URLs as logically linked if:
    - one is a “subdirectory” of another, or
    - they are within N degrees of each other in the Web’s link graph
  - Compared their approach (multiple-page algorithm) to baseline (single-page algorithm) by having human subjects in various fields run queries and judge results
    - MAX_LEVEL and MAX_LINK: parameters that they didn’t vary but should have

Sentiment Identification Using Maximum Entropy Analysis of Movie Reviews

- Mehra, Khandelwal, and Patel
  - Used movie reviews from `rec.arts.movies.reviews`
  - Got people to rank their preferences for various movies on a website (but only had six users!)
  - Implemented personalized classification: based on a user’s movie preferences, used maximum entropy model to classify reviews to find ones that they would like…? I’m not even sure what they did.

News Meta-Search Across Multiple Languages

- Patel
  - Built “Global Reporter” system that tried to implement CLIR for news articles
  - Used Babel Fish to translate both queries and articles
  - Evaluation: six users issued nine queries each using a) English-only and b) multi-language and judged relevance of results

Parametric Search Using In-memory Auxiliary Index

- Verman and Ravela
  - Problem: traditional parametric search is slow because of disk accesses necessitated by frequent database reads
  - Solution: since metadata is relatively small compared to corpus itself, store in main memory
  - Used Lucene, MySQL, Citeseer Postscript docs with associated metadata
More comments based on examples

- If your algorithms crucially depend on certain parameters, vary them.
- Make your write-up clear!
- If you’re using human subjects to evaluate your system, you really should try to get a statistically significant sample.