Final Project

36-350, Fall 2013

You will work on projects in groups of 2–4. You will upload a text file to Blackboard with your proposed group, and your rankings of at least 3 projects, by 11:59 pm on Thursday 14 November.

If you do not wish to propose a group, you will be assigned one; I also reserve the right to adjust groups. (Likewise, if you do not state preferences, preferences will be assigned to you.) If you have an idea for a project not on this list, you are welcome to propose it, but please do so before the deadline.

There will be three components to the project: an oral presentation during the final examination period, (approximately 15 minutes long), a written report describing the problem and what you did to solve it, and (documented) code.

1. *Drunken chessmaster* Imagine playing chess with only one piece (a knight, rook, queen, or bishop, ...) by always selecting your next move at random from all possible legal moves. This is something a drunken chessmaster might do. How many moves would it take for your piece to return to where it started? Does it matter where you start? If you play this way for a very long time and then stop which square will you most likely land on? You will write code to simulate this process, to estimate averages and distributions of answers to these questions and others like it.

2. *Stepwise model selection* In regression problems that often come up in data mining, there are very many covariates $X_1, \ldots, X_p$ that are potentially related with the response Y. An important problem is determining which subset of variables are the best predictors of Y. When p is large, using all p variables may be a bad idea; it is also computationally intractable to try every possible subset of the p-variables. Instead one may try fitting a sequence of models in a greedy manner by starting from 0 variables and then adding one variable at a time by choosing the "best" variable at each step according to some criterion. Alternatively, start with a full model using all p variables and then remove the "worst" variable at each step. We will provide you with data consisting of a vector of responses $\mathbf{Y}$ and a matrix of variables $\mathbf{X}$. You will write code to fit a sequence of linear regression models using ordinary least squares and some criterion from adding/removing variables to the model. Your code will also perform cross-validation to estimate the mean squared prediction error of each candidate model and to choose the best one.

3. *Word frequency* A classical claim in quantitative linguistics is **Zipf's law**,

which says that the number of words from the dictionary which appear k or more times in a large text or collection of texts is proportional to $k^{-\alpha}$: a few words appear a very large number of times, a vast number of words appear only once. you will get a canonical literary text in electronic form; you will write code to estimate α from the text, plot the fitted distribution and the data, and assess uncertainty and goodness of fit.

4. *Document classification* You will get news stories from the New York *Times* with known subject classification. You will write code to extract the text, turn the word frequencies into features, and automatically sort the text into categories, using regression-like methods. We will provide texts for two groups, one learning to discriminate articles about art from articles about music, the other learning to discriminate news stories from editorials.

5. *Markov chain language models* A simple statistical model of language is that words follow a Markov chain, with the next word being independent of earlier words given the latest word. You will write code to fit this model to a collection of documents, to simulate new text from it, and to compare it to a second-order Markov chain, where the next word is independent of earlier words given the latest *two* words.

6. *Markov chain genetic models* DNA consists of a series of distinct "base" molecules, conventionally written A, C, G, T. The sequence of bases specifies the genetic information used to grow organisms, such as yourselves. A simple statistical model of DNA is that bases follow a Markov chain, with the next base being independent of earlier bases given the latest base. You will write code to fit this model to the genome of a real organism (the social slime mold *Dictyostelium discoideum*), to simulate new DNA from it, and to compare it to k^{th}-order Markov chains, where the next base is independent of earlier bases given the latest k bases.

7. *High-dimensional two-sample tests* The Kolmogorov-Smirnov test (`ks.test` in R) is a simple but powerful procedure for testing whether two samples come from the same distribution. It only works for one-dimensional data. You will write software to take two *multi-dimensional* data sets, randomly project them down to one-dimensional distributions, and apply the K-S test to their projections. You will need to take into account multiple testing issues.

8. *Income Distributions of the Rich and Anonymous* The "World Top Income Database" (`http://topincomes.g-mond.parisschoolofeconomics.eu`) provides data on the very highest income taxpayers, across countries and across multiple years. Many economic models say that the upper tail of the income distribution should follow a Pareto distribution, with the number of people whose income exceeds x being $\propto x^{-\alpha}$, but do not say what α should be (other than > 1). You will obtain the data, estimate

α over countries and years, and produce visual comparisons of trends in inequality over time and space.

9. *Red State, Blue State, Rich State, Poor State* Over the last few decades, states with a higher average income level have tended to vote more for the Democratic Party in presidential elections, and poor states for the Republican Party. Within each state, however, lower-income individuals have tended to be more likely to vote for the Democrats, and higher-income people to vote Republican. Using data (which will be supplied) from the 2008 National Election Survey (the most recent available), fit logistic regression models summarizing how voting choices depend on individual income, using split/apply/combine to analyze each state. Examine how the state-by-state income coefficients vary with state-level characteristics.

10. *Earthquake intensities* Using data `http://www.quake.geo.berkeley.edu/anss/catalog-search.html`, fit time-varying intensity models for earthquakes in California, Italy, and Japan, since 1970. Determine if variation in intensity in each region exceeds what would be expected under a simple constant-intensity Poisson process.

11. *Locality-sensitive hashing* "Hash functions" are functions that take in complicated data objects and return numbers, which are generally compressed (and not unique) IDs or labels for the objects. They have many uses in programming, like checking whether files have been changed, or lists which store and retrieve information by value rather than by position. "Locality-sensitive hash functions" are special hash functions which are designed so that similar objects will, with high probability, get similar hashed values. If we start with a complex object, like a document or a sound file, and want to find all of the other objects in a large collection which are similar to it, comparing the target object to all the others in the collection is very slow; locality-sensitive hashing is used to limit our search to promising candidate objects, ones which are likely to be close to the target. This project will implement locality-sensitive hashing, from scratch, for documents.