High-Dimensional Statistical Analysis

Group Members:
Stephen McDonald (sam6@princeton.edu)
Brenhin Keller (cbkeller@princeton.edu)

Project Repository: hdstats-framework

Overview:

Our project involved developing a useful interface for high-dimensional data analysis. We worked to knit together the various aspects of data analysis: data cleaning, performing analysis, running diagnostics on the analysis, etc. In general, many of these individual components are implemented well in libraries/modules for a variety of languages such as R, Matlab, and Python. However, we were not aware of any convenient integrated package which stitches together all of the components of high-dimensional data analysis. Building such an interface was the primary goal of this project. Our final project covers only a subset of the potential techniques to implement, however we believe it was successfully able to develop an easily extensible framework for data analysis.

Very broadly, we break the data analysis process into the following major components: data importation, data cleaning, data analysis, and model diagnostics.

Data Importation:

The first and simplest step in the analysis of any dataset. We have implemented options to import data and uncertainty matrices from delimited ASCII files and from binary files of known structure.

Data Cleaning and Transformation:

Automated data cleaning has become an increasingly crucial first step in the analysis of multidimensional data the volume of available data grows. While the ideal method of data cleaning is specific to the sources and assumptions of a given dataset, a number of basic techniques are broadly useful for arbitrary data. In general, we view data cleaning as a way to remove outliers, identify possibly erroneous data, and make initial transformations of the data to prepare it for analysis.

The following options are implemented in the data cleaning menu:

Technique	Description
Outlier removal by standard deviation	Replace with NaNs any data points a specified number of standard deviations above or below the mean.

Outlier removal by percentile	Replace with NaNs any data points a specified number of standard deviations above or below the mean with.
Replace NaNs	Allows for the interpolation or deletion of NaN data.
De-mean	Set the mean of each variable to zero
Normalize variance	Divide each variable by its variance, normalizing variance to 1
Bootstrap resample	Resample the dataset n times with replacement, each time adding Gaussian noise with variance determined by the specified 2-sigma error for each sample. Error can be set for each sample in the import menu, and will default to 5% relative error

Data Analysis:

The actual analysis desired by end users is likely to involve a number of sequential components. The user may want to use some technique to reduce the dimensionality of the data, then perform a regression on the reduced data. Alternatively the user may be interested in clustering the data into major groups and performing a separate regression analysis within each group. Towards that end, we implemented a variety of algorithms according to 4 major groups: Regression, Classification, Dimension Reduction, and Clustering. We did not implement all of the available algorithms in scikit-learn, rather we developed a framework to make it easy to add additional analysis techniques.

Regression:

Algorithm	Short Description
OLS	Basic ordinary least squares regression
Lasso	Regularized regression. Penalizes the L1 norm of the coefficients, leading to sparse models.
Ridge	Regularized regression. Penalizes the L2 norm of the coefficients.
Elastic Net	Combination of L1 and L2 norm penalties
OMP	Orthogonal Matching Pursuit. Targets the number of desired nonzero coefficients and uses estimations of basis functions to fit them.

LARS	Least-angle regression. Efficient method for high-dimensional situations.

Classification:

Algorithm	Short Description
Logistic Regression	Standard logistic regression for classification.
LDA	Linear Discriminant Analysis assumes the underlying classes come from a multivariate Gaussian distribution and same the same covariance matrix
QDA	Quadratic Discriminant Analysis assumes the underlying classes come from a multivariate Gaussian distribution, but fits separate covariance matrices for each class.

Dimensionality reduction

Algorithm	Short Description
PCA	Principal Component Analysis. Converts variables to a set of orthogonal components in order of decreasing importance
Randomized PCA	Variant of PCA that uses randomized SVA for speed
Sparse PCA	Variant of PCA that favors sparse principal components
ICA	Independent Component Analysis. Similar to PCA, except components need not be orthogonal
Isomap Embedding	Nonlinear dimensionality reduction using Isomap (isometric mapping) Embedding
LLE	Nonlinear dimensionality reduction using Locally Linear Embedding
Spectral Embedding	Nonlinear dimensionality reduction using Spectral Embedding

Clustering:

Algorithm	Short Description

K-Means	The standard clustering algorithm. Assigns clusters such that each point belongs to the cluster with the nearest mean.
MiniBatch K-Means	A variant of K-Means clustering with better scalability.
Mean Shift Clustering	Finds clusters with means in areas of high density, but penalizes clusters that are too close together. Does not require you to know the number of clusters *a priori.*
Spectral Clustering	Clustering on the normalized laplacian. Good for non-convex clusters
DBSCAN	Density-Based Spatial Clustering of Applications with Noise. Like Mean Shift, identifies clusters by density and does not need the number of clusters as an input. Starts from point of high sample density and expands cluster bounds outwards.

Model Checks/Diagnostics:
Whenever performing data analysis, it is vital that diagnostics are performed on the output of the analysis to make sure it was appropriate. This may include checking the assumptions of the model (for example, are the residuals from an OLS regression normally distributed?), plotting the output of the model (the various clusters for a clustering algorithm), or checking that the data is amenable to the analysis at all (is the data matrix singular?). These checks are quite dependent on the analysis performed. Regression techniques, particularly OLS, have many testable assumptions. Clustering techniques on the other hand, are more of an exploratory technique which can rarely be tested. For a full list of the checks performed for any particular technique, see the project documentation. The table below gives the statistical tests employed. All of these are derived classes from the “check” class and are located in checks.py.
For a technical review of the checks, see the documentation.

User Guide:

Necessary dependencies:

Python
The program requires the user have Python installed. It was built using Python 2.7, and no testing has been done to guarantee performance on any other version of Python.

R
As some of the code is implemented via a python interface to R, it is necessary to have R installed for that code to run. It was tested on a relatively old version of R (2.15) but should work on newer versions as well.

rpy2

Rpy2 is an interface between Python and R. Essentially, it allows Python to run an interactive R environment, and data can be passed back and forth between R and Python. The program was built using rpy2 2.3.0.

Using the Program:

To run the program, navigate to the hdstats-framework directory. Enter the following command at the command prompt:

```
$ python hdstats-framework.py
```

You will then be presented with a menu of options. In general, the options are self explantory. You will almost certainly want to import data first, as all of the analysis and visualizations are dependent on having some data to work with. We will now walk through a few examples on datasets we have loaded in so you can see how the different analysis sections work.

Regression Example:

We will walk through a basic sparse regression problem.
Navigate to the hdstats-framework directory. Start the program:
python hdstats-framework.py
Select `1: Import`
Select `1: Import Data Array`
Select `1: Delimited ASCII`
Enter: `sparseReg.csv`
Enter: `,`
Enter: `1`

The data has now been successfully imported.
Select `0: Exit`
Select `3: Analysis`
Select `1: Regression`
Select `2: Lasso`
Enter: `.0005` for the alpha parameter
The code will then run, providing the coefficients, the R-squared, most likely a warning that the lasso algorithm did not converge, and plots of the residuals and regularization path plots.

Implementation:

The project was written entirely in Python, with some R code implemented in Python through rpy2. This was done to utilize statistical tests that are available in R but not Python.

The primary Python packages used were Scikit-Learn, for machine learning algorithms, and Statsmodels, for statistical tests. Pandas, Scipy, and Numpy were also used

somewhat for their array/dataframe capabilities. Python was chosen to be our language of choice, because we wanted to implement from an existing library of algorithms, and Python's Scikit library has a much greater variety of well developed and well documented algorithms than we believe exists in other languages like C++, FORTRAN, etc. In the future, extensions of this project could involve writing certain algorithms in a fast compiled language and then using existing tools to interface with the main Python program. Our program structure should be well designed to handle such extensions.

Interface:
The user interface was implemented entirely on the command line, with a textual menu-based interface using simple text IO. Upon calling hdstats-framework.py, the user is presented with the following menu

```
Select task:
  1. Import
  2. Data cleaning
  3. Analysis
  4. Visualization
  5. Enter Python interpreter
  0. Exit
```

A menu item is selected by typing the number of the desired option, followed by the return key. After selecting a specific method through one of the subsequent sub-menus, the user will be prompted for any required input. While simplistic, this interface is highly portable and largely self-explanatory. In order to alleviate the somewhat constraining nature of this interface, an option is provided to enter a built-in python interpreter, from which the program variables and classes are accessible.

Testing:
Unit tests were implemented using the standard Python unittest package. The testing is not comprehensive, but rather was created to check particular outputs as seemed wise. In general, the each test checks some portion of the output from implementing a particular algorithm on a small dataset against a known solution, either analytically or obtained from R. Of course, since we are using algorithms implemented by others as part of major Python libraries, we are fairly confident in general that the algorithms themselves are accurate. Our tests are provide both an extra check regarding the accuracy of the underlying algorithms, as well as a check on our internal storage of the analysis outputs. For example, many errors in our code came from storing what we thought was a vector of estimated model coefficients, but was in fact an array, containing the coefficients and associated p-values, or some other output. Our unittest framework is meant to try to catch some of these errors, if in the future either the underlying techniques we are using change their output format, or if we change our internal framework for a class in a way that incorrectly stores the output.

The tests can be run by running the command: python projectTests.py

We did additional ad hoc testing as necessary, and to test the format of our command line interface we walked through the different steps manually. An area for improvement of our project would be to find a way to automate these manual checks.

Version Control:
We used Github for version control. Overall, we found it very straightforward to use github for collaboration. We have slightly over 200 commits, fairly equally shared between both group members. Since we were typically working on different files, we rarely had merge conflicts. Conflicts occurred most often in the dataAnalysis.py module, which is the driver for the Analysis section, by taking the users input and selecting the desired analysis type. Conflicts there were relatively minor, and never required any significant time or concern to be fixed.

The github repository for the project can be found at:
https://github.com/brenhinkeller/hdstats-framework

Documentation:
Documentation of the code was done using doxygen, along with the doxypy package meant to help integrate doxygen's special commands into Python's docstrings. Doxypy helped make many of the special commands available, however there were still some limitations. Doxygen seems to stress highly documented code, and we made a deliberate design choice to follow this practice of documenting most member functions and variables. This fairly complete documentation did come at the cost of somewhat more cluttered code, especially due to some of the limitations of doxypy. The html pages for the documentation are included on github.

Further Extensions:
There are several major areas in which we could extend this project. Of course, the more straightforward one would be to add additional data cleaning, analysis, checks, and visualizations. We implemented only a fraction of the techniques available on scikit-learn, so there is plenty of room to extend in that direction.

The use of rpy2 in our project has also opened the possibility of more fully leveraging R's capabilities. Currently, we only use rpy2 to use R statistical tests for which there wasn't a corresponding test available in Python. However, an interesting avenue for improvement would be to do a major comparison of the capabilities of R and Python, and then build the program to implement each algorithm or statistical test in the language in which it is best suited. Given our program design, this should be fairly straightforward to implement. An analysis class, or statistical check, that would best be implemented in R could keep its Python class structure, but internally use rpy2 to do the analysis in R. The user would not need to know the different conventions, data formats, or even syntax of R or Python- all they would need to know is the type of analysis they want to run, and the program could handle the rest. From our personal

experience, this would be very beneficial, since it can be quite challenging trying to do some work in one language, then trying to determine the proper format to put the data in when transferring to another language.