LECTURE 1: CLASS POLICIES, TOOLS AND TECHNOLOGIES

Week of 8/30	Lecture Notes
Content	class policies, class tools, introduction, what this course is about, data mining
ExpectedOutcomes	• overview of course policies • overview of data mining concepts, algorithms,
Readings &Supplemental	**REQUIRED**» 2014. Zaki, Mohammed J and Meira Jr, Wagner; *Data minin*
Homework	**DUE:** Monday, 9/4 - midnightPlease see the Blackboard/Github repo for wha

---

NOTES

Welcome to the course! Over the coming weeks we're going to have a lot of fun, learn a lot about data mining tools, technique and algorithms and end up prepared and excited to do analyses and explorations of our own!

INTRODUCTION TO DATA MINING

At the core of data mining is the exploration of *finding meaningful patterns in data*, often in pursuit of patterns that will help explain a phenomenon that would ordinarily not be possible with human effort alone. Data mining borrows much of its heritage from applied mathematics and statistical thinking, but with the rise of *massive data*, traditional statistical techniques often fall short and require completely different algorithmic approaches and computational technologies.

This *massive data* cannot be overstated. When we look at even a small susbset of the "dataverse" (a term I am loosely using to encompass the whole of data), data from the web and social media, some estimates suggest that data is being created at a rate never before seen. For example every minute:

- over 300 hours of YouTube video is being uploaded,
- 77,000+ hours of video are being streamed over Netflix,
- 51,000+ apps are downloaded from the Apple iTunes store,

and so on. Our ability to consume this data, let alone make sense of it is indeed being challenged, hence the rapid rise of data mining, machine learning and data science technologies in the last decade.

Data mining is a discipline unto itself, but it can easily be considered a subset of the broader area of *data science* which encompasses machine learning, data analytics, data visualization, data engineering, etc. Data mining, is a common thread throughout all of data science, and as such the topics we cover will represent a broad swath of nearly all of what is now termed "data science".

One common summarization of "data science" has been given by Drew Conway that suggests data science is the intersection of *software engineering*, *mathematical modeling and statistical analysis* applied over a particular *domain* or set of complementary domains, with **data** being at the foundation of it all.

Our attention will largely be on the *statistical and mathematical*, *software engineering* components, and we will add to this *data engineering*, *data visualization*, with an eye towards specific areas of *text mining*, *social network analysis* and *data ethics*.

While some argue the term "data mining" might be subsumed by "data science" or that it is a bit of a misnomer, we'll stick with the term, knowing that while we may "mine" for "data", the outcomes of such mining should be carefully considered in the context of whether such outcomes are meaningful or not. We will come to understand that many results and outcomes of data analyses must be couched carefully as to avoid misinterpretation, since often there may be outcomes that confirm what we *want* rather than what *actually exists*, or even worse something that *doesn't* exist but rather produces enough signal to seem meaningful, but in reality is just noise. This is no different than the criticisms lodged at statistical models that confirm the relationships being sought, rather than the actual relationships that truly describe a phenomenon.

DATA MINING WORKFLOW

The data mining process can be summarized as follows:

DATA ENGINEERING AND PREPARATION

Data cleaning
Data integration
Data selection
Data transformation

DATA EXPLORATION, MODEL BUILDING AND ANALYSIS

Statistical analysis
Pattern mining

Prediction
Classification
Clustering

EVALUATION AND MODEL INTERPRETATION

Pattern evaluation
Model interpretation and evaluation

PRESENTATION

Visualization and Reporting
Data narratives

You might look at the process and think that most of your time will be spent in analysis, but this is often not the case. As we'll explore soon data engineering and preparation can *often be the most time consuming component* of the process, since the type of data (structured, unstructured), the source of the data (database, online, etc.) and the volume of data might require significant cleaning, selection and transformation. Luckily, there are many tools that can ease the complexity of these tasks.

DATA SOURCES

Data mining can be carried out over a number of different data sources, the most common of which are :

- relational databases
- data warehouses
- transactional databases
- object, temporal, time-series and sequence databases
- real-time data streams

Within an enterprise, there are many sources of data that may be the target of a data mining exploration. For example, some businesses keep extensive relational databases on products, draw from transactional databases for retail sales data or tap large data warehouses to get segments of data required to answer more nuanced questions of importance to a business need.

Take for example a company that stores customer profiles but may want to fully understand how their customer demographics play into the sales of specific products. They might be interested in knowing if certain products are associated with certain income brackets, etc. Such analysis may be easy to perform if customer income is known (or can be drawn on from a data warehouse), otherwise such an analysis may require data from another source external to the company.

Internal data sources, are an important source of data and will often be a *primary* source of data when embarking on an analysis, but *external data sources* are often equally important for data mining tasks. For example, the US Census Bureau keeps extensive data on the demographics of the US and economy. A company may want to correlate data from the Census, for example, with their own data or supplement data they have with data from a government source, since such sources tend to be statistically reliable and consistent. These data may be accessed directly or stored locally and updated when necessary.

External sources are plentiful, and many are *open* (e.g. most if not all public government data), yet some external data may come at a cost, for example, sports data or data from companies that collect data of a particular kind (social media, web analytics, etc.). For this course, we will work with open data sources, keeping in mind that some data may not be so freely available.

DATA MINING FUNCTIONALITIES

We will focus our attention on at least five major data mining functionalities listed in the table below. Each entry in the table includes information about the typical scenarios these functionalities appear in.

Functionality	Scenario
Class summarization and description	Determining the characteristic of a class or group in
Frequent patterns, association and correlations	Determining the patterns that are commonly associa
Clustering	Uncovering patterns in data that are typically not kn
Classification and Prediction	Assigning classes or making predictions given some p
Outlier analysis	Discerning data that may appear to be outside the ty

TOOLS

There are three primary tools we will used in this course:

- Github: we will use this for our course materials, in conjunction with Blackboard.

- Python: Python is one of a few languages widely used in data mining, machine learning and data science. It has easy syntax and powerful expressiveness, and is backed by a strong ecosystem of tools and libraries for the entire data mining workflow.

- Jupyter Notebooks: Jupyter Notebooks (also knows and "ipynb"'s"or"nb"'s) are an important tool in analysis, visualization and narrative building (reporting). They are quickly beecoming the *de facto* tool for building and distributing data analysis code.

Github

Github is a system / web application / platform that provides a beautiful way to store, view, manage, share and revise code. While the primary content on Github is *running* software, whether that be in C, Java, HTML, Python or the many hundreds of other languages in popular use today, it can also be a place for traditional text files (data, papers) and other binary data (Word/Excel documents, images, etc.) Github's strength is in the ability for it to facilitate software collaboration, and today it is the premier place on the web for large scale open-source, collaborative software projects. Though open and public projects are Github's forte, it provides the ability for you to host private projects and allows you to make the decision later whether it is appropriate to make such projects public or not.

Github is built on top of what is called **git**, which is a **revision control system**. The core concept behind such systems is that they manage your text-based code files and allow you to keep track of the changes (revisions) to those files. In a system like **git** you can invite others to work on your code as well, which can allow more than one person to make contributions to the code. What is even more useful is that such changes can be tracked across a project so that those contributions and changes can be seen, reviewed, modified and otherwise managed. Thus, when someone (even you) makes a change to a file (or files) and makes those changes known to the revision control system, others with access to those files can not only see those changes, but integrate them into their own.

It provides a great deal of functionality which might appear to be complex and overwhelming. Most of the functionality you will need in the basic daily use case is very narrow, so don't be discouraged by the depth and complexity of git's documentation.

What's Git all about?

In revision control systems your code is typically stored in a **repository** and that repository often lives and is managed on a remote server. This is done for a

variety of reasons, one if which is to provide others access to your code. Git has the advantage of such code and its revisions being controlled locally and only when you're ready can those changes optionally be put onto the remote server. The relationship between Git and Github is that Github provides a nice visual shell over Git and also serves as the remote git server, so you don't have to think about it when you want to share your code with others (or have a remote copy of it).

You will always need to install git on your local system, and don't ever *have to have* a remote server, but if that is the case, the code in such a local project will never be seen by, or syncronized with (backed up on) a remote server, thus making it difficult to (though not technically impossible) share with others in the general case.

Projects to Repositorities

In git your code lives in a repository. This is simply the location/directory for your files and it lives (initially) local to your file system on your computer. You can create a repository anywhere on your file system, but it is often a good practice to keep git repositories in a common location when it makes sense to do so. Furthermore, there is no limit to the number of repositories or the number of files under the control of a repository.

You can consider a repository as a project containing a single focus of interest. For example, you might think if your repository as containing a single complete software application you're building. Similarly, you might think of it as a place for all the files of a single course over a semester, or a single topic of interest. Your concept of "project" isn't really restricted, though there are some best practices.

In Github, as in git, the anchor of a project is the repository.

Resources

- Introductory videos on git
- Introductory guide on Github

Python

Python is among a few of the *de facto* languages used in data mining, machine learning and data science broadly.

It has a simple, expressive syntax that is praised for its consistency and similarity to *psuedocode*, indeed, the syntax can be learned in a short time (usually a day or

two) and because it is an interpreted language, beginners can be more productive because of the instant feedback and lack of complex compilation requirements of other languages. Though Python is fantastic and widely loved, it is not without its criticisms, one of which being the complexity of its library ecosystem and the dual fork of the language (Python2 and Python3).

We will focus on Python3 as that is where the future of the language lives and also that support for Python2 is quickly waning, with official support ending by the decade's conclusion.

Instead of spending any time here going over Python, you'll be diverted to several good online resources for learning Python.

The Python Data Science Stack

Python has a well developed ecosystem for doing data mining, machine learning and data science. In fact, the language is well suited for many text manipulation tasks such as tokenizing, parsing and general data munging – in just a few lines of code you can process a large text file, extracting from it a large corpus for post-processing.

There are five primary tools (in addition to Python) we'll use throughout our data mining journey, Numpy, Pandas, SciPy and Scikit-Learn summarized in the table below.

Tool	Summary
Python	we'll focus on Python 3
NumPy	the primary library used for high performance data matrix operations
Pandas	the high level library of data structures used to make working with data a joyful exp
Scikit-Learn	the core machine learning and data mining library
SciPy	the core scientific and numerical computing library, which we'll restrict our use to it
Jupter Notebooks/Lab	the primary tool for building code and data narratives see below

Anaconda

While you can certainly install each of these separately (and it is perfectly OK to do so), it is advised that you breathe easier by installing this stack via Anaconda, which is becoming the *de facto* packaging of the Python Data Science Stack. It includes hundreds of the most popular Python libraries (including those listed above) and greatly simplifies the update, upgrade and dependency management required of all the packages in the distribution.

Jupyter Notebooks

What?

In the last few years, Jupyter Notebooks have become one of several standard tools being used in data analysis and data science today. We will make heavy use of its features, but will begin with the basics. So what is a Jupyter Notebook?

A Jupyter Notebook can be minimally summarized as :

- an *interactive*, *executable* **document**, that
- *mixes code* and *data* with **narrative** (text, technical notes, etc.), that
- is *sharable* and *editable*.

Think of your notebook as a way to explore data, while sharing the code along side your data exploration, while simultaneously having a conversation (through the narrative) describing the code, the data and the goal(s) you're trying to accomplish.

Jupyter is much more than described above, and certainly not the only tool out there, but it is one of the best at what it does and is getting better with each new version!

Why?

In the scientific method, we often like to break the process down into the following rough steps:

1. formulate a question (from some general observation(s) / phenomenon)
2. propose a hypothesis that might explain the phenomenon
3. develop a prediction or model
4. gather data and test predictions / model
5. refine, alter, reject, expand hypothesis (and if necessary model)
6. **[iterate between steps 3-5]**
7. develop general theories from the outcomes

The scientific method works best when there is supporting documentation of each of these steps, because when sharing the outcomes of research, we are very much concerned with making sure our work is *repeatable* and *reproducible*. Work is *repeatable* when we execute the same methods on the same data and get the same result. It is *reproducible* when we execute the same methods on new or different data and get results that are consistent with our theory or hypothesis. Both are necessary for good science.

To make the "science" in "data science" relevant we will use the scientific method as our anchor. The power of Jupyter Notebooks is demonstrated in their flexibility to support nearly all computational and data-driven phases of the scientific method and hence data science. We can use the documentation and narrative building tools of the notebook to document our questions and hypotheses (1 and 2 above). We can even link the documentation of our hypothesis to the original research being referenced or built upon. Using the interactive programming paradigm of Jupyter (along side the documentation) we can test computations and build predictions and models in code, and even test early versions of those models interactively, perhaps even holding on to variations during iterative refinement.

How?

Data-driven analyses today can be considered a combination of explanatory narrative with computational support for that narrative. With technology like Jupyter, we can marry the two so that they can co-exist together where they (arguably) make sense to.

Jupyter supports narratives by introducing the concept of a "cell", where a cell *is a container for stuff* ... that stuff being one of the following

- executing code (in a language like Python),
- plain text and Markdown,
- math via \LaTeX mixed in with plain text and Markdown, or
- HTML.

What makes Jupyter different from other environments like Matlab is that cells containing executing code can be in a variety of supported languages. While Python, R and Julia have robust support in Jupyter, other languages like Java, Javascript, FORTRAN and others can be included in a notebook. Furthermore, a cell can support dynamic visualizations in HTML and Javascript, taking interactivity to another level.

Plain text narratives with Markdown

Jupyter cells that are designed for text can use a simple markup notation called Markdown and also \LaTeX for mathematical notation. Markdown is much like plain old text with a few simple notations for things like bold, italics, headings and linking. In general, Markdown is not hard to learn and the more you use it the more natural it becomes. The motivating philosophy behind it was to strive for readability, simplicity and unobtrusiveness in writing documentation.

We won't spend a lot of time here with Markdown as there are superb tutorials online to get you on your way.

Math narratives with \LaTeX

\LaTeX has been used for over three decades to produce highly professional scientific documents and books. It is likely you own at least one textbook (or course notes) that were written using \LaTeX, especially if the subject is math, physics or computer science. One reason you may not have learned to use it, however, is that in general it has a high learning curve for beginners, and while the final output of a \LaTeX document is most certainly going to please, it may have taken much longer to produce your first version when compared to Google or Microsoft tools.

Luckily, Jupyter makes this much easier by removing the need for any complex formatting, and instead just supporting Markdown, while at the same time providing a great deal of support for the bulk of moderately complex \LaTeX math without much of the fancy typesetting that brings beginners to cut their losses and quit before a final product can be realized. Breath easy, this will not be as difficult as may seem.

If you need to build narratives that are math heavy, Jupyter support for basic mathematical notation is built in to help you build nice looking documents (with *beautifully* typeset math) that could *theoretically* be exported to PDF and look almost as nice as a document written in a word processing environment like Google Docs or Microsoft Word (or even full \LaTeX). While admittedly lacking many of the features of a "word processor", for computational narratives the Jupyter environment is compelling and the Jupyter Lab product promises even greater features and integrations.

Mathematical narratives in Jupyter can be useful in many contexts, for example,

- to explain a theoretical result or show the usage of a specific mathematical technique,
- to provide a mathematical proof of a result that you are deriving,

- to show the mathematical underpinnings of an algorithm,
- to show the formulae used in a concrete calculation or manipulation of data.

Here are a few math examples (view the source code for this notebook to see how easily it can be done):

Suppose that,
$$
f'(x) = { \lim_{ \Delta x \rightarrow 0 } { { f(x + \Delta x) - f(x) } \over \Delta x } }
$$

and maybe
$$
\int x^n dx = { {x^{n+1}} \over n+1 }
$$
therefore we have
$$
\sum_{k}^\infty { {t^k \over k!} = e^t }
$$

and in rare cases
$$
\sum_{\substack{
0

When?

Jupyter Notebooks can be used in a variety of contexts, only a few of which are mentioned below:

- documenting homework,
- documenting an algorithm or research methodology,
- writing a paper that is about code or a dataset,
- developing and documenting a research idea that explores a theoretical result backed by an implementation (code),
- exploring and documenting a dataset or an algorithm on a specific dataset,
- testing code,

- playing with code,
- and many other contexts.

Resources

The best place to begin learning how to use Jupyter is to go directly to

- the Jupyter.org site,
- but for inspiration see the gallery of Jupyter examples
- and some for Earth Sciences notebooks examples,
- and another general gallery,
- and many, many more you may find and explore on your own ...