W205: Storing and Retrieving Data
Spring 2015

Craigslist Price Comparison

Final Project

Instructor: Alex Milowski

Team Members:
Nate Black | Arthur Mak | Malini Mittal | Marguerite Oneto

April 28, 2015

Table of Contents

1 Introduction 4

1.1 The Problem: Comparing Prices 4

1.2 Application: A Price Comparison Tool 4

2 Project Architecture 4

2.1 Architecture Diagram 4

2.2 Data Process Overview 6

2.2.1 Data Source and Acquisition 6

2.2.2 Data Cleaning 6

2.2.3 Data Storage 6

2.2.4 Data Retrieval 7

2.3 Data Pipelines 7

2.4 Data Representation 8

2.4.1 Raw and Cleaned Data in S3 8

2.4.2 Production Data in MongoDB 9

3 Implementation 10

3.1 Project Tools and Libraries 10

3.2 Acquisition and Storage 11

3.2.1 Scraping 11

3.2.2 Example Raw JSON entry 12

3.3 Cleaning 12

3.3.1 Exploratory Analysis 13

3.3.2 Explanation of the Classification Problem 13

3.3.3 Supervised Learning 13

3.3.4 Unsupervised Learning 14

3.3.5 Cleaned JSON entry 15

3.4 Retrieval 16

3.4.1 Retrieval from S3 to MongoDB 16

3.4.2 Retrieval from MongoDB via User Query on Website 17

4 Results 19
4.1 Website Features 19
4.1.1 Fair Market Value 19
4.1.2 Price Trend 19
4.1.3 Price Comparison Across Cities 19
4.1.4 Price Search 20
4.2 Data - The 3 V's 20
5 Challenges 20
5.1 Scraping Data 20
5.2 Setting up mrjob/EMR 20
5.3 Classifying iPhones 20
5.4 Retrieving Data 21
6 Future Work 21
6.1 More Items 21
6.2 More Cities and Countries 22
6.3 Price Index 22
6.4 Arbitrage Across Geographies 22
6.5 API 22
6.6 Add Sources 22
Appendix 23
A1: Website Access and Github 23
A2: Website Screenshots 24

1 Introduction

Craigslist is a classified advertisements website that was founded in the late 1990s. Over the past decade, the website has expanded to over 2,000 markets in 50 countries. The website rapidly gained popularity due to it's low cost, ease of use, and user freedom. With services in housing, personal advertisements, jobs, for sale, and community, the website has a plethora of data. But sifting through that data can be extremely challenging due to the free-form nature of the website.

1.1 The Problem: Comparing Prices

Fair value is defined as the rational estimate of the market price for a good. But how does one obtain fair value?[1] In particular, how does a user know an advertisement on Craigslist is fairly valued? People commonly, and often unknowingly, attempt to determine the fair value of a good by comparing prices across different markets and against historical prices. While Craigslist is wonderful for its low cost and ease of use, it is both difficult as well as time-consuming to use Craigslist as a price comparison tool. This project attempts to filter the massive corpus of Craigslist data into a human-readable summary of prices across various markets over time.

1.2 Application: A Price Comparison Tool

The goal of this project is to produce a user-friendly web application that provides a price comparison interface when a user conducts a search for a particular item.[2] The website will be able to answer questions such as:

1. What is the fair market value of the item?
2. What is the price trend for a given city?
3. What is the average price across the country?

2 Project Architecture

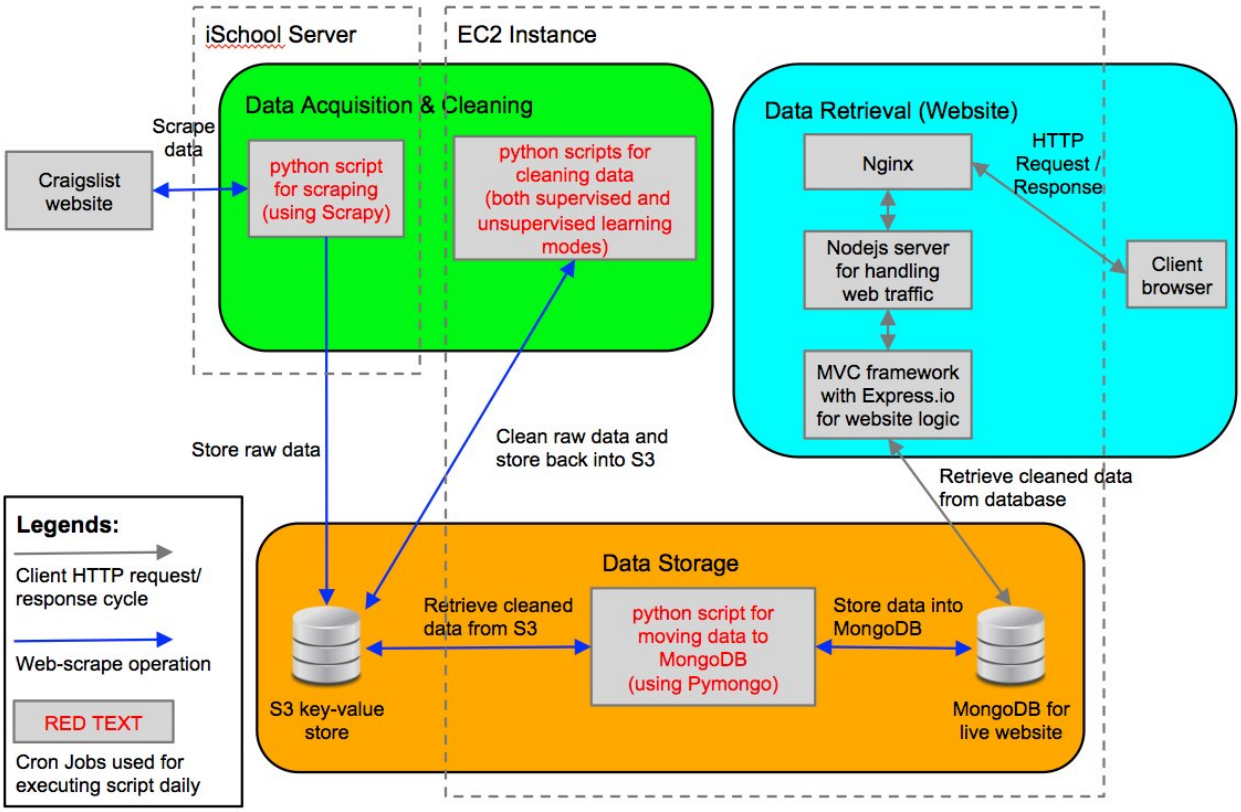
This section describes the idealized architecture of our project.

2.1 Architecture Diagram

Below, we provide a visual representation of the project's complete architecture:

[1] http://web.stanford.edu/~jdlevin/Papers/PriceSearch.pdf

[2] Please see the Appendix for screenshots of the final website.



The project consists of three core components. The green area represents the data acquisition and cleaning component. Cron Jobs are used to run Python scripts daily. The first Python script uses Scrapy, a web crawler framework, to crawl the Craigslist website and store the raw data in Amazon Simple Storage Service (S3). Two Python scripts retrieve the raw data from S3 and clean the data in-memory, one for supervised learning and another for unsupervised learning, before putting the data back into the S3 store as cleaned data. The script to acquire data from Craigslist is running on the UC Berkeley iSchool server because Craigslist blocks the data retrieval process from all IP addresses of Amazon Elastic Cloud Compute (EC2) instances. The other scripts reside on EC2 instances.

The orange area represents the data storage component. The S3 store previously mentioned is part of this component. Another Cron Job is used to transfer the cleaned data daily from the S3 store into the MongoDB, which is the live database. Note that the Python script and MongoDB reside within the same EC2 instance, whereas S3 is outside of the EC2 instance.

The blue area represents the data retrieval component. The website allows users to retrieve and visualize the required data. As the user interacts and queries the data via the website, the user/client's browser generates HTTP requests that pass to the Nginx web server. Nginx handles such web traffic by passing the request to the Nodejs server. This Nginx and Nodejs setup is a commonly used reverse proxy setup for better website security and performance. The Nodejs server then processes the HTTP request via an MVC (model-view-controller) framework that controls all the routing and website logic depending on the user's request. Express.io was used

as the framework. The Express.io framework includes all the website logic, such as determining the data query from MongoDB and rendering the HTML based on the returned data. After the HTTP request is processed, the Nodejs server and Nginx can pass an HTTP response back to the client browser so that the user can see the result on his or her web browser.

2.2 Data Process Overview

2.2.1 Data Source and Acquisition

We are using Apple's iPhone as the item of choice for this project. The data was sourced from the 23 largest Craigslist U.S. markets. As the Craigslist.org website does not provide an API to retrieve the listings, raw data was scraped from the website on a daily basis and stored in S3. Custom web scrapers were developed for each city to obtain data in an automated fashion while respecting the website's HTTP request traffic (i.e. data was scraped slowly).

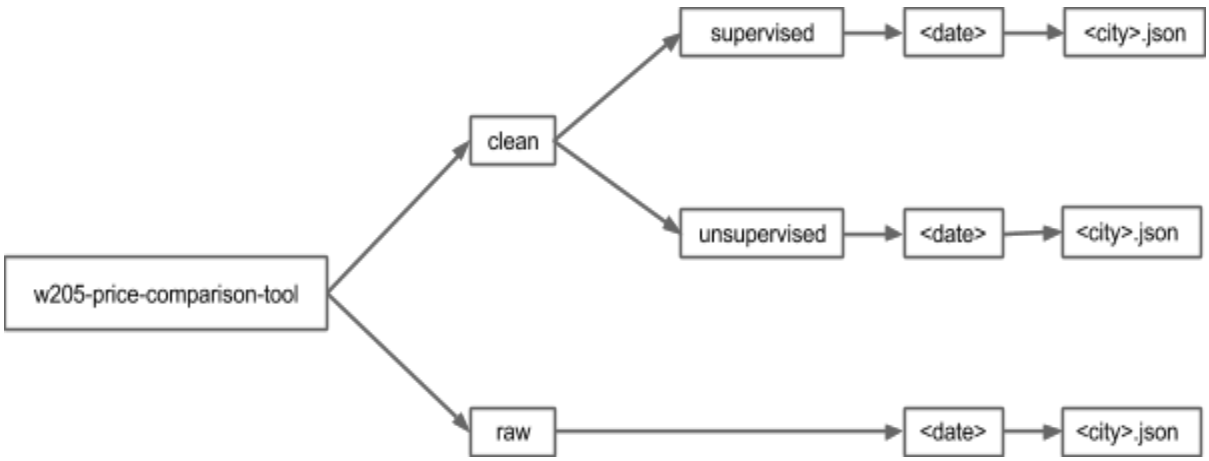
2.2.2 Data Cleaning

The acquired data needs to be cleaned before use in the live environment of the web application. Unneeded fields are removed, certain fields are re-formatted (e.g. dates), missing values are excluded, and filters are applied. Natural language processing (NLP), both supervised and unsupervised learning methodologies, was utilized to differentiate between various types of iPhones (1/3/4/4s/5/5c/5s/6/6+). The scrubbed data was stored back in S3.

2.2.3 Data Storage

Raw data from the data acquisition process is stored in Amazon S3. After the data is acquired and properly cleaned by the Python scripts, it is stored back into S3. Another Python script is used to pass the data from S3 to our database of choice, MongoDB, which can be used by the live web application.

The following figure shows how the JSON files are organized on S3:



The relevant names are as follows:

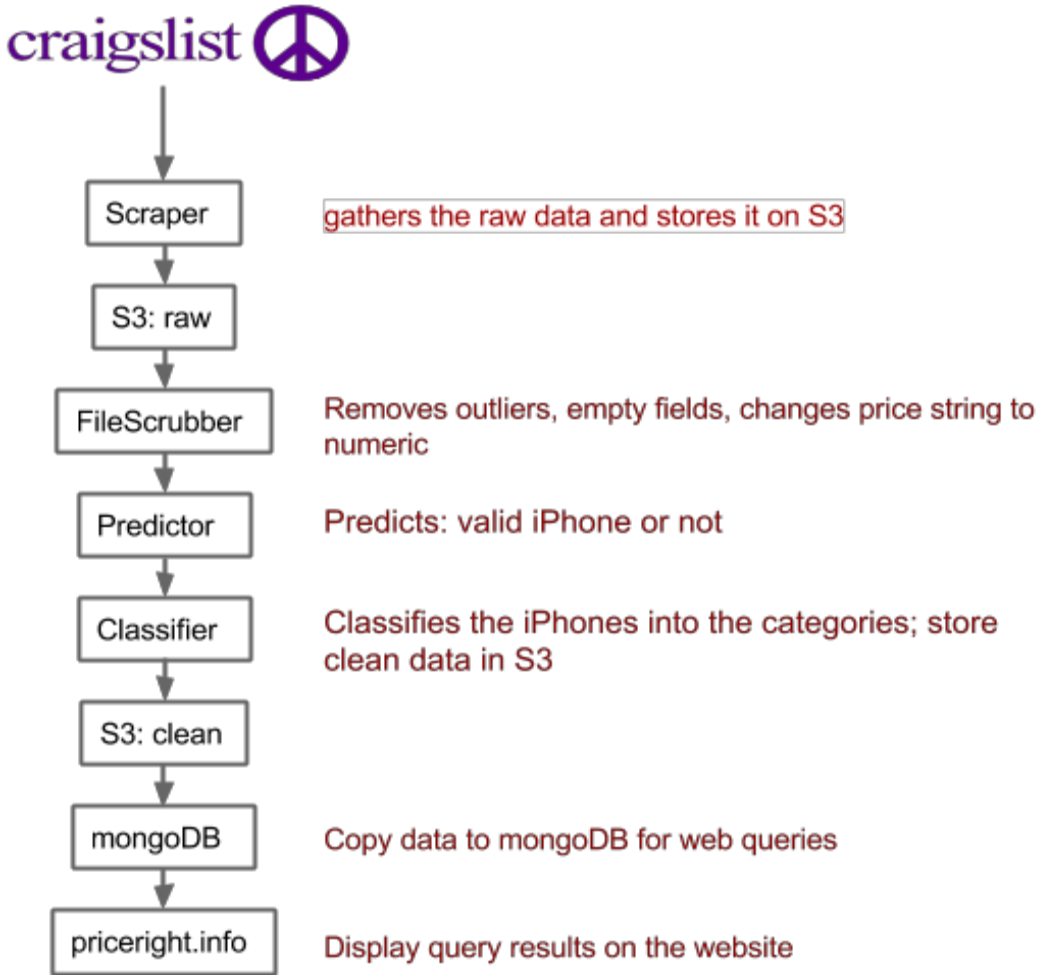
- Bucket name: ‘w205-price-comparison-tool’
- Sample key for raw data: ‘raw/2015-04-21/sfbay.json’
- Sample key for supervised clean data: ‘clean/supervised/2015-04-21/sfbay.json’
- Sample key for unsupervised clean data: ‘clean/unsupervised/2015-04-21/sfbay.json’

2.2.4 Data Retrieval

The web application is a live website where the user can search for a given iPhone in a given location over a specified timeframe. Data is stored across 8 collections in MongoDB. As users interact with the website, queries are executed against the database and results are rendered in real-time.

2.3 Data Pipelines

The following figure describes the flow of data from the source to the website:



2.4 Data Representation

2.4.1 Raw and Cleaned Data in S3

Data is scraped from the Craigslist website (HTML) and stored as JavaScript Object Notation (JSON) objects in S3. JSON objects were chosen because of their compatibility across systems (S3, Python, and JavaScript), flexibility, and because MongoDB natively stores data in the binary JSON (BSON) format, which is just a binary representation of JSON. Other formats (e.g. comma-separated values) could have been used in lieu of JSONs, but it was convenient to keep a consistent format across the data pipeline.

The following table lists the attributes, their data types, and example values of a raw listing stored as a JSON object:

Name	**Value**	**Type**
category	“cell phones”	String
create_date	“2015-03-17 15:53”	String
title	“IPHONE 5-64GB-VERIZON UNLOCKED-Mint condition”	String
url	“http://atlanta.craigslist.org/atl/mob/4936499733.html”	String
country	“US”	String
price	$280”	String
area	“atlanta”	String
state	“GA”	String
location	“(ATL)”	String
subarea	“all atlanta”	String

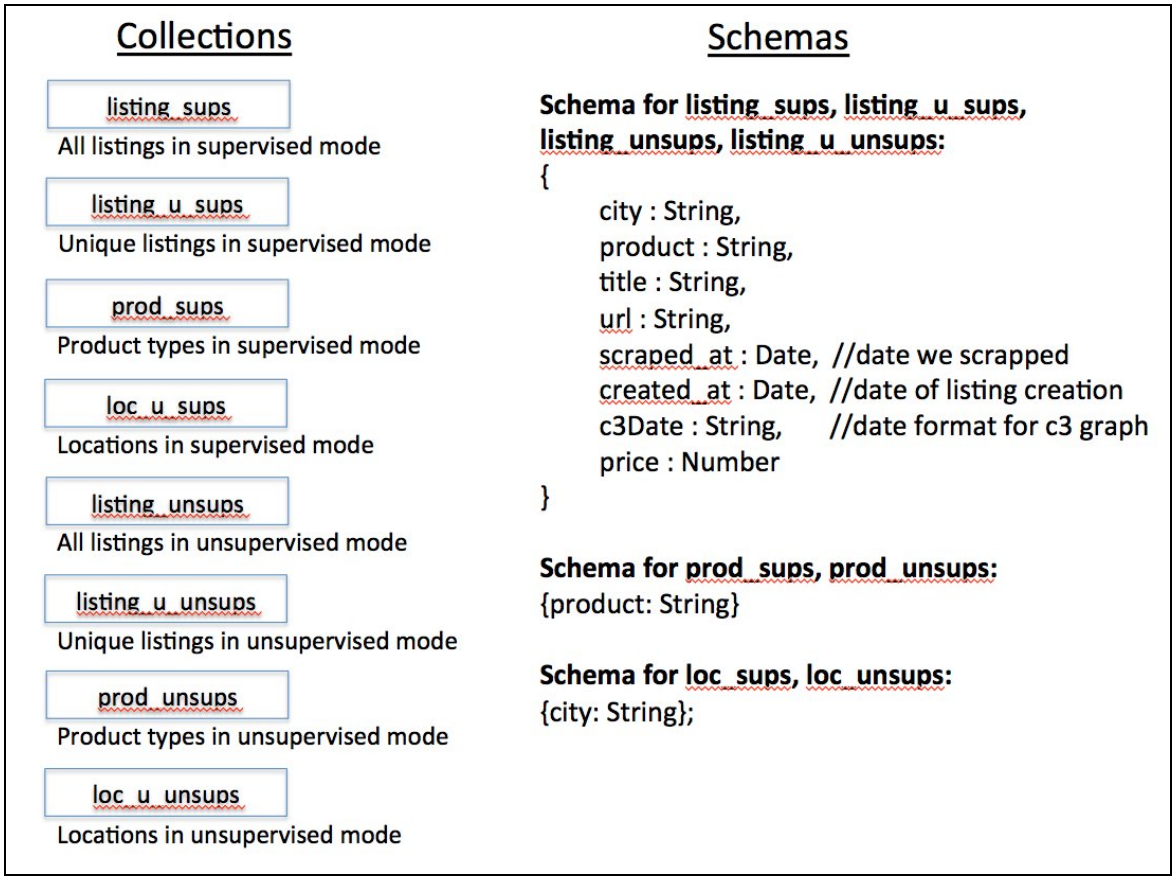
The following table lists the attributes, their data types, and example values of a cleaned listing stored as a JSON object:

Name	**Value**	**Type**
area	“atlanta”	String
category	“cell phones”	String
country	“US”	String

create_date	“2015-03-17T00:00:00.000Z”	String - ISO format
location	“(ATL)”	String
price	280.0	Float
state	“GA”	String
subarea	“all atlanta”	String
title	“IPHONE 5-64GB-VERIZON UNLOCKED-Mint condition”	String
url	“http://atlanta.craigslist.org/atl/mob/4936499733.html”	String
pred_bin	1	Binary
pred_class	“5c”	String (categorical, one of 0, 1, 3, 3g, 4, 4s, 5, 5c, 5s, 6, 6+)
scraped_date	"2015-03-31T00:00:00.000Z"	String - ISO format

2.4.2 Production Data in MongoDB

S3 data is passed into MongoDB on a daily basis via Cron Job. The data is split into 8 different collections. The collections and schemas are shown below. Note that we did not use foreign keys. Therefore, the collections do not have links to each other.



3 Implementation

This section describes the actual experience of implementing the idealized architecture above.

3.1 Project Tools And Libraries

The following list highlights many of the tools and libraries that were utilized as part of the project.

- Amazon S3 - Used to store the raw data and scrubbed data
- MongoDB - Used to store the production data
- Amazon EC2 - Houses the MongoDB database, and two Cron Jobs
- boto - Used to access S3 from Python
- Pymongo - Used to access MongoDB from Python
- Gensim - Python package utilized for unsupervised text classification
- nltk - Python package for natural language processing
- Sci-kit Learn - Python package utilized for supervised text classification
- Pandas - Python package for efficiently munging data
- matplotlib - Python package for visualizing data
- Scipy - Python package for scientific computing
- Numpy - Python package for its vectorized operations and speed

- re - Python package for regular expressions
- JSON - Python package for JSON objects
- cPickle - Python package to serialize the training data
- Scrapy - Python web crawling framework
- iSchool Server - Host of our web scraping script
- mrjob - Python package to implement map reduce code
- Cron Jobs - running 4 separate daily python scripts on linux machines / instances
- Node.js - Javascript library for server-side interactions
- Mongoose.js - Javascript library for data modeling between express and MongoDB
- Angular.js - Javascript library for data-binding on client browser
- Express.io - Javascript library for web application MVC framework
- C3.js and D3.js - Javascript libraries for visualizations
- jQuery.js - Javascript library for enhanced cross-browser functionalities
- Twitter Bootstrap - Front-end framework for responsive website layout

3.2 Acquisition and Storage

3.2.1 Scraping

To scrape the data, the Python packages Beautiful Soup and lxml were initially used to explore the process. With an initial prototype, more robust solutions were explored. The Python web crawling framework Scrapy was deemed the most appropriate option. Scrapy was chosen as it is a complete framework to crawl the web pages, while with Beautiful Soup, crawling needs to be manually simulated. Scrapy provides most of the fundamental needs of a crawler, along with various variables that can be set to customize the spider according to the needs of the project.

With a crawler successfully implemented in one city, the spider framework was replicated for the 23 major cities that Craigslist displays on the website. The decision to replicate code was mainly to keep the code simpler, to have the ability to run the spiders for different cities on different machines, and also to be able to run these spiders in a parallel manner, giving us the potential to scale up to many more cities/countries in the future.

Once these scrapers were running, another roadblock was discovered: Craigslist blocks IP addresses if it detects bot activity. Several solutions were attempted to get past this issue:

- Running the scraper in an “autothrottle” mode which automatically tries to adjust the crawler to the optimum crawling speed
- Using a download delay, in addition to the autothrottle, which adds a delay between each successive request
- Using a set of commonly used user-agent strings, randomized so that each request to the website uses a randomized user-agent
- Running the scrapers on multiple EC2 servers, which did not work as the IP addresses used by these servers are blocked en masse by Craigslist.org
- Randomizing the order in which the cities are scraped
- Randomizing the amount of download delay used

- As a final step to prevent being blocked, the “text” field was removed from the data, as this one field was increasing the total number of requests by a factor of 100 and was unneeded for classification.

These steps allowed scraping to be conducted in a consistent, repeatable manner. The process has now been productionalized on the iSchool server and scrapes data daily via a Cron Job.

3.2.2 Example Raw JSON Entry

We store the results in JavaScript Object Notation (JSON), as shown in the example structure below:


```
{
        "category": "cell phones",
        "create_date": "2015-03-17 15:11",
        "title": "iPhone 4S Verizon, Apple warranty",
        "url": "http://atlanta.craigslist.org/atl/mob/4897242808.html",
        "country": "US",
        "price": "$125",
        "area": "atlanta",
        "state": "GA",
        "location": "",
        "text": "\n        I have an excellent condition black iPhone 4S, 16GB for Verizon.",
        "subarea": "all atlanta"
}
```


This data provides the needed information about the phone, price, location, category for possible future use, date the posting was created, etc. This data is used to answer the questions that we have outlined above.

3.3 Cleaning

The acquired data needed to be cleaned before use in the live environment of the web application. All the unwanted, unnecessary fields were removed. Fields were re-formatted as datetime objects and numbers where applicable. We also filtered the data based on price ranges $60-$1,500, as there are outlier values that have a large impact on the results. Items with missing prices were also removed.

In order to differentiate between cases where a search for a certain iPhone keyword brings up exact, related, as well as unrelated items, we used natural language processing (NLP) techniques to match potentially different descriptions of the iPhone to the same item. In particular, we used both supervised and unsupervised learning techniques to categorize the postings into either an iPhone 1, 3, 4, 4s, 5, 5c, 5s, 6, or 6+ bucket. On the website, the user will be able to choose to see the results using either method.

The MapReduce programming paradigm was used to parallelize the cleaning of the 23 raw files. A map-only task was created that performed the filtering and classifying described above.

3.3.1 Exploratory Analysis

An exploratory data analysis was conducted to assess the scraped data. The analysis was conducted on one day's scraped data across all cities. Approximately 40,000 postings were analyzed. The analysis highlighted several potential data issues:

- Outlier values exist that have a large impact on the results (e.g. iPhones priced at $5,000 or iPhone accessories for $25)
- Data should be filtered based on price ranges $60-$1,500
- Missing values exist for a small population of items where a price is not specified, and they would be excluded from the data set
- Prices are stored as strings and will need to be manipulated to floating point numbers
- Invalid entries (NaNs) exist and will need to be filtered
- Some posts contain multiple phones for sale
- Posts contain certain stop words such as "Trade", "Repair", etc.

3.3.2 Explanation of the Classification Problem

On our website, the user can choose to see the prices of the iPhone by model type. They can look for an iPhone 3, 4, 4s, etc.

The titles of Craigslist postings do not yield such clean categories. We needed a way to take a title like "❤▫❤▫ IPAD IPHONE 5 5s 6 6plus (((6″ and turn it into a category like "5s." This example shows other roadblocks that needed to be overcome. Some listings included non-iPhone items (iPad). Others listed multiple iPhone models for sale (5, 5s, 6, 6+). Such listings needed to be removed to prevent them from skewing the average price calculations.

We turned to natural language processing to classify the items from their titles. Several different methods were tried, both using supervised and unsupervised learning. Supervised learning allowed transparent models to be built. Unsupervised learning was less transparent, but provided the opportunity to scale the project to include items beyond the iPhone.

3.3.3 Supervised Learning

The supervised learning classification was a two-step process: a machine learning classifier and regular expressions. The initial classification was binary (iPhone/not iPhone) using the Naive Bayes algorithm. Other supervised learning techniques were considered such as Logistic Regression and Support Vector Machines, but the Naive Bayes algorithm was chosen for its simplicity and transparency. The model was trained on a random sample of 2,000 postings. The specified model accurately classified iPhones in 97.5% of the training set. Prior to specifying the model, text is stripped of punctuation and stop words. In the second step of the classification process, regular expression techniques were used to classify iPhones into various categories. The regular expressions were transparent, tractable, and provided favorable results.

The supervised learning classification utilized several popular Python packages that are commonly used for data analysis and machine learning. Pandas, NumPy, and Sci-kit Learn were

the primary packages used. These packages were chosen because they are well-documented and generally considered the industry standard for machine learning in Python.

3.3.4 Unsupervised Learning

An unsupervised learning algorithm was also utilized to classify Craigslist postings into different item categories. The unsupervised learning model would allow the project to easily expand outside the realm of just iPhones (i.e. scale). Unsupervised learning can categorize postings without having to manually train the model. Unlike the supervised model where the model is trained once, the unsupervised model is "online" in that it is trained daily as new data is added.

The unsupervised learning algorithm employs Latent Semantic Indexing (LSI) and K-means clustering. The algorithm takes the following steps:

- Remove punctuation, common stop words, custom stop words, and words that only appear once from the posting titles
- Tokenize the posting titles
- Create a dictionary and transform the corpus of Craigslist titles into bag-of-words vectors
- Transform the bag-of-words vectors using TF-IDF (Term Frequency-Inverse Document Frequency) weightings
- Transform the TF-IDF vectors representing the corpus into a latent N-dimensional vector space using Latent Semantic Indexing, where N is the number of topics chosen.
- Use a K-means clustering algorithm to create clusters of the LSI vectors that represent the Craigslist titles
- Use each cluster's centroid vector to label each cluster by topic (e.g. iPhone 4, iPhone 4s, etc.)
- Use regular expressions to distinguish between categories with ambiguous tags

In the end, the LSI-Clustering algorithm was only able to classify two-thirds of the Craigslist postings. One-third of the titles were too complex for the algorithm to parse. But on the remaining two-thirds, the model was able to classify iPhones into their correct categories 99% of the time. This was sufficient for our purposes of calculating an average price for each iPhone model, without any extraneous items falling into the wrong categories.

In building the LSI-Clustering algorithm, we ran into several additional problems.

The algorithm was able to process some complex titles. But these titles were always thrown into the same bucket together, no matter the iPhone model. For example, the title "iPhone 6 plus still in the box, with all accessories" was placed in the same bucket as "iPhone 5c, blue, with otterbox case. All offers considered." Furthermore, this bucket was combined with one of the standard iPhone model buckets, and the model bucket to which it was added changed randomly from day to day. One day, the complex titles would be placed in the iPhone 4s bucket. The next day, they would be placed in the iPhone 6+ bucket. The solution to this problem was to use regular expressions to weed the complex titles out of that day's bucket. This practice, which is so

specific to iPhones, technically makes the final LSI-Clustering algorithm non-scalable to new items.

Before regular expressions were used, though, another method was tried to see if it could process the complex titles more effectively. This was Similarity Indexing. It performed much better than the LSI-Clustering algorithm, but this method required training on the specific iPhone model types (4/4s/5/5s/5c/6/6+), which also rendered it non-scalable.

Finally, the algorithm was not able to distinguish between an iPhone 6 and an iPhone 6+. This required some translation of the data before processing. For example, the listings “iPhone 6 PLUS,” “iPhone 6 Plus,” and “iPhone 6 plus,” were all changed to read “iPhone 6+.” This, again, is so iPhone-specific that it makes the algorithm non-scalable to new items.

The final LSI-Clustering unsupervised learning algorithm uses the following Python packages:

- Gensim - Used for fast Vector Space Modeling, with applications to topic modeling, document indexing, and similarity retrieval with large corpora.
- NLTK - Used its stop words list to remove common words from the Craigslist titles.
- Scipy - Used to transform Python lists into arrays for k-means clustering. Also used to conduct the k-means clustering itself.

3.3.5 Cleaned JSON Entry

The cleaned data is stored as a JSON object as follows:


```
{
        “area”: “atlanta”,
        “category”: “cell phones”,
        “country”: “US”,
        “create_date”: “2015-03-30T00:00:00.000Z”,
        “location”: “”,
        “price”: 170.0,
        “state”: “GA”,
        “subarea”: “all atlanta”
        “title”: “iPhone 4S Verizon, Apple warranty”,
        “url”: “http://atlanta.craigslist.org/atl/mob/4897242808.html”,
        "pred_bin": 1,
        “pred_class”: “5s”,
         “scraped_date”: “2015-03-31T00:00:00.000Z”
}
```


The price is now a numeric value. There are three additional fields:

- pred_bin: A binary value which predicts whether the listing is a valid iPhone or not
- pred_class: The predicted class of the iPhone (1, 3, 3g, 4, 4s, …)
- scraped_date: The date on which this listing was scraped

3.4 Retrieval

There are two main steps in the data retrieval process. The first step is to load the data from S3 to the production MongoDB database. The second step is to load data onto a website upon user's request. There is a master script that can run on any new EC2 instance that sets up all required functionality for both steps of the data retrieval process. The process includes updating the server, installing Python, installing MongoDB, installing the Node server, installing the Nginx server, installing Git, pulling the entire website repository from GitHub, configuring Nginx and Node, as well as ensuring the Node and Nginx servers keep respawning even when the EC2 instance dies.

3.4.1 Retrieval from S3 to MongoDB

There is a script to load data inside the EC2 instance. The Python boto library is used for retrieving the cleaned data from S3. The pymongo library is used for interacting with the MongoDB database. During the data load process, there is a multi-function Python script that can operate in two main modes: load all data from S3 or just load the incremental data from today. With such a setup, there is the flexibility to load all data when the server is first setup. Then, a Cron Job can be used to add incremental data to MongoDB each day.

The data load script works in several steps. First, the data load script needs to connect to S3 and iterate through every listing over a specified time period (either all data or just today's data). Then, for each listing, price and date information need to be converted to the proper variable types since S3 stores all values in the form of strings. The listing information is then pushed into a collection of the MongoDB. Every time we load data into MongoDB, it is done in a batch of 10,000 listings, because inserting data into MongoDB is much faster in batch.

After the listings are loaded into MongoDB, there are many duplicate postings in the data. The reason is that when the data is scraped each day, there are both old and new listings of Craigslist posts. The old posts would be duplicated as the scraper would keep getting those posts each day until Craigslist removes them. This duplication of data is necessary as one of the goals of the project is to obtain the average price of all listings for each day. A section of the website, however, only shows unique listings and requires the removal of duplicate listings. Therefore, an "aggregate" function in MongoDB was utilized to group the data by URL link of the product in order to get the unique listings. This de-duplicated set of listing data is then stored in another MongoDB collection. One collection contains all listings with both old and new posts, whereas the other collection is just the unique listings. While the entire project could have been completed without creating the unique listing collection, the additional collection allows for greater clarity and significant improvements in the website's data retrieval.

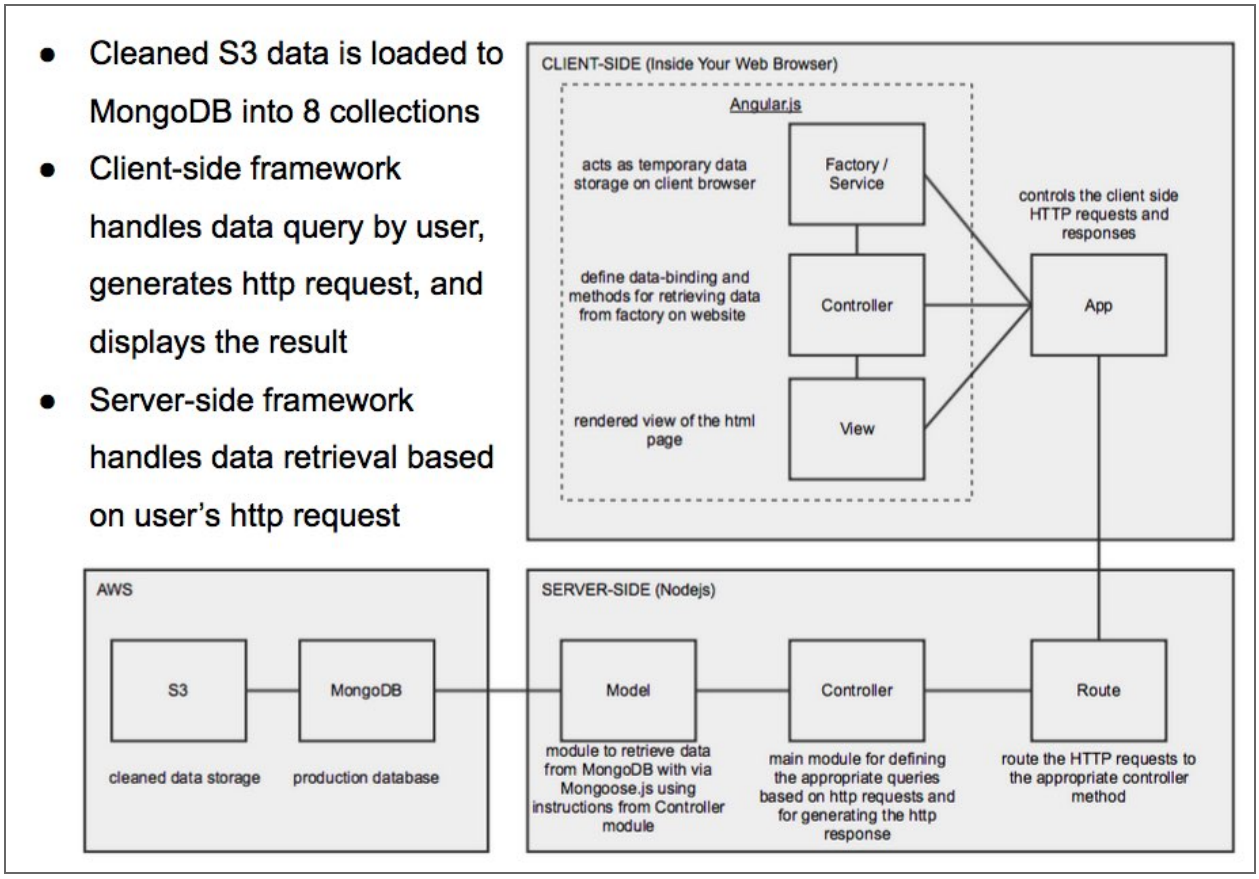
After both collections are properly loaded into MongoDB, two additional queries are used to determine the unique products (eg. iPhone 4s, iPhone 5, iPhone 6, etc) and locations ("SF Bay Area", "Atlanta", "Austin", etc). These values are options for users to query on the website. The results are then stored in separate collections.

In summary, there are 4 collections: one for full listings, one for unique listings, one for products, and one for locations. Since the data was cleaned using both supervised and unsupervised learning techniques, the collections need to be replicated. Thus, in the end there are 8 collections in MongoDB as follows:

- listing_sups - all listings for supervised mode data
- listing_u_sups - unique listings for supervised mode data
- prod_sups - unique product types for supervised mode data
- loc_sups - unique location types for supervised mode data
- listing_unsups - all listings for unsupervised mode data
- listing_u_unsups - unique listings for unsupervised mode data
- prod_unsups - unique product types for unsupervised mode data
- loc_unsups - unique location types for unsupervised mode data

3.4.2 Retrieval from MongoDB via User Query on Website

The web application is a live website where the user can search for a given iPhone in a given location over a specified timeframe. The chosen stack for building the website is commonly known as the “MEAN” stack (MongoDB, Express.io, Angular.js, Node.js). This is a full- stack where the server, database, and model-view-controller (MVC) framework can be setup such that the website can work interactively with MongoDB. Whenever a user queries the website, the “model” component of the MVC framework will send a request to acquire the needed data from the MongoDB database stored on Amazon EC2. The “controller” component of the framework will perform any further calculations and data preparations prior to passing the “view” component of the framework. The user will then be able to see the data in the form of a rendered D3.js visualization. There is actually slightly more complexity in our MVC framework as we incorporated Angular.js for our website where we must define both client-side and server-side frameworks. The retrieval process of our project can be represented by the diagram below, which illustrates the MVC framework implementation:



The main benefit of the setup is that it allows a high level of control of how data is being handled from both the client and the server side. At the heart of the data retrieval process is the controller on the server-side. It contains 6 main MongoDB query methods, which are listed below: (please refer to Appendix A2 for which of these methods get activated when user interacts with our website)

- getLocations() - get the the unique location type
- getProducts() - get the unique product type
- getNumResults() - get the count of queried results
- getListings_priceTime() - get the average prices over a given time period (sorted by chronological date of data scraping)
- getListings_priceCity() - get the average prices of different cities (sorted by alphabetical order of city name)
- getListings_allItems() - get the list of list of unique postings (sorted by date of posting on Craigslist)

The resulting website allows for the following types of input queries:

- Use data from supervised or unsupervised learning modes
- Which product
- Which location
- Which period

The website then retrieves and displays the following types of output based on the input queries:

- Latest average price
- Product average price per day
- Product average price per location
- Listing of unique products of the given period

In terms of the listing of unique products on the website, 2000 product listings of the queried results would be shown. The filtering function allows the user to type a keyword for instant search among listings. The filtering is almost instantaneous due to the benefit of storing the data directly onto the web browser using Angular.js (i.e. no need to do an additional MongoDB query while filtering the results). The sorting function has two modes: “Sort by Latest Posting Date” and “Sort by Lowest Price”. Each time the user changes the sort, a new query will be passed to the MongoDB to retrieve 2000 product listings.

4 Results

The project achieved its objective of providing Craigslist shoppers with a user-friendly interface for comparing prices. Users can now answer the questions:

1. What is the fair market value of the item?
2. What is the price trend for a given city?
3. What is the average price across the country?

4.1 Website Features

For a visual representation of the website features described below, please see Appendix A2.

4.1.1 Fair Market Value

Craigslist shoppers can now find the fair market value of any model of iPhone in their city. Once they choose their model and city, the most recent day’s average price is displayed on the website. The shopper can compare this average to the price given on a particular iPhone posting to ascertain whether they should buy it, given the iPhone’s condition and any other considerations they may have.

4.1.2 Price Trend

If shoppers would like to know if the price of their chosen iPhone is going to go up or down in the future, they can turn to the historical price trend graph for their city. Shoppers are able to enter the start date and end date for which they would like to see the trend. This could help them determine whether they might want to wait to buy, if they see that the price trend is going down.

4.1.3 Price Comparison Across Cities

Shoppers who are looking for the best deal may also want to see if another city may have cheaper prices than their own local Craigslist is showing. This information is displayed in a bar chart on the website. It reflects the most recent day’s average price for each of the 23 cities. The price differentials shown in the bar chart might just make up for the shipping costs between cities.

4.1.4 Price Search

Once shoppers know the average price, they may want to start comparing listings directly. They can do this at the bottom of the website, where data on 2000 Craigslist postings is displayed. The data contains the iPhone model being sold, the title of the posting, the advertised price, the city, and the date the item was posted. The data can be filtered using any keyword. For example, if a shopper is looking for an iPhone 5s, he can type "5s" into the filtering text box, and 2000 listings for iPhone 5s's will appear. The listings can also be sorted by price and by posting date. And each listing has a URL link that allows the user to go and shop directly on the Craigslist website.

4.2 Data: The 3 V's

Volume: The project scrapes and stores about 13MB of raw data per day. For six weeks, this is approximately 550MB. The clean data makes up about 1.2GB data, including data cleaned via both the learning modes. There are close to 2 million listings that are saved in the MongoDB. Any future scaling toward more data and more geographic locations will increase the data.

Variety: The free-form nature of Craigslist postings adds variety to the data, which necessitates the use of NLP methods. Anticipated scaling would result in additional variety as new items are added.

Velocity: We did not encounter a big velocity of data, as the project was scraping the website once a day only.

5 Challenges

5.1 Scraping Data

As mentioned earlier, the main challenge in scraping the data being IP blocking by the Craigslist.org website, we needed to try out various tactics to create realistic HTTP requests. Another issue that we faced was that for some reason Scrapy was not able to write directly to S3 any file greater than 1MB in size. The workaround was to store them locally and then transfer them to S3. Additionally, the scraper stops after scraping 2400 items, even if there were more listed on the website. We have not been able to figure this one out yet.

5.2 Setting up mrjob/EMR

Running mrjob locally was easy. But getting the mrjob to run on EMR required a lot of fiddling with the various options in the configuration file. In addition, debugging the script when running on EMR was not easy. The map/reduce job took less than a minute to run locally on the EC2 instance as compared to about 15 mins on EMR; this was probably due to the time taken to instantiate multiple instances of EC2 every time.

5.3 Classifying iPhones

While building the unsupervised learning algorithm to classify iPhones by model, we used the

Gensim Python library. In particular, we used its LSI modeling function to reduce the dimensionality of the weighted word vectors. We chose to model on 100 topics, and the output of the LSI model should have been 100-element vectors. But the LSI function was inconsistent and would produce some vectors shorter than 100 elements. In general, the function would output approximately 50 to 60 short vectors out of 25,000.

This inconsistent performance was very hard to detect. The LSI vectors are passed to the K-means clustering algorithm, and the short vectors would most often, but not always, crash the program at that step. We assumed the problem was with the K-means clustering and proceeded to spend much time debugging that algorithm. We never suspected that the Gensim function could perform inconsistently, often producing bad output.

Once the bad output was detected, we proceeded to try Latent Dirichlet Allocation (LDA) in place of LSI. The LDA model was consistent, but it performed worse than the LSI mode at classification, so we returned to using LSI. To solve the inconsistent performance problem, we now remove the 50 to 60 Craigslist postings that produce short LSI vectors before they are passed to the clustering algorithm.

5.4 Retrieving Data

We encountered four interesting obstacles as part of this data retrieval process. First, when listings exceed one million, there is insufficient default allocated memory within MongoDB to perform the “aggregation” function for grouping the listings into unique ones. MongoDB was manually adjusted to use additional server memory so the query could run properly. Second, after listings were aggregated, the document output is larger than the maximum limit of 16MB defined by MongoDB. The query was modified such that it pipes the aggregation output directly into the unique listings collection in order to overcome the maximum document size limit issue. Third, while Angular.js allows us to make the web browser act like a database, we do run into issues when we try to store too much listings data onto the web browser. This is because data-binding from Angular.js requires a lot of memory from the client browser. Our web browser simply ran out of memory and froze. Therefore, we set the browser to store an upper limit of 2000 product listings to ensure optimal filtering and sorting performance.

Beyond issues related to volume of data, we also had to overcome issues with web operation as we need to have 4 different Cron jobs being run at different times: one for scraping data, one for cleaning data in supervised NLP mode, one for cleaning data in unsupervised NLP mode, and one for loading data from S3 into MongoDB. We managed to make the Cron jobs run at different times each day. We would need to put more work into ensuring the robustness and exception handling of this setup in the event that one of the Cron job scripts breaks down.

6 Future Work

The project could be extended in several ways.

6.1 More Items

The first goal would be to add more items whose average prices and price trends would be calculated for the Craigslist shopper. We would eventually hope to provide coverage for all 34 of

Craigslist's "For Sale" categories, from antiques and appliances to tools and video games. Another category we would like to tackle is housing, especially providing basic pricing information for rentals based on number of bedrooms, number of bathrooms, and square footage.

6.2 More Cities and Countries

Another extension would be to add more US cities, as well as different countries, to the price comparison tool. Craigslist exists in over 700 cities in 70 countries around the world.

6.3 Price Index

Calculating a Consumer Price Index (CPI) has always been a controversial topic. In the US, the Bureau of Labor Statistics (BLS) is responsible for generating the CPI on a monthly basis. It has changed its formulas over time due to intense criticism. This CPI is used to determine the rate of inflation, which has wide-ranging impacts, from determining eligibility for government assistance (social security, food stamps) to valuing assets in the financial markets.

One application of collecting prices across many different cities would be to create a "Craigslist Consumer Price Index." It would be interesting to see how much difference there is between the CL CPI and the BLS CPI over time.

Price indices could also be created for each country as well. These could then be compared across countries to be used in purchasing power parity and currency exchange rate calculations.

6.4 Arbitrage Across Geographies

The current Craigslist Price Comparison Tool already provides a comparison of prices across cities, allowing shoppers to see if they can buy their iPhone for a lower price on another Craigslist site. We would like to enhance the user interface to make the arbitrage opportunities more explicit and easier to find.

6.5 API

This project could be extended to create a Scrapy-like framework. Adding a new city/product would require minimal subclassing of the API, and the framework would handle all the details with respect to the actual scraping of data, cleaning the data, the NLP classification, etc.

6.6 Add Sources

There are many other classified advertisement websites like Craigslist, such as eBay, Backpage, and Geebo. Adding listings from these websites would increase the value and usefulness of our price comparison tool for online shoppers, making it easier for them to find the right deal.

Appendix

A1: Website Access and Github

The access link for the website is:

www.priceright.info

The Github repository (private) for our project is:

https://github.com/maktrix16/w205_priceright

(Please see the screenshot of the website on the next page)

A2: Website Screenshots