CarDealz Design Specification

Version 2.3

CarDealz Prediction Systems
A Machine Learning Prediction Application

Supervisor:
Darragh O'Brein

Development Team:
David O'Regan

22 May 2014

CARDEALZ

Simple, yet eligant prediction

Website:
http://cardealz.strikingly.com/#home

Blog:
http://blogs.computing.dcu.ie/wordpress/oregand7/

Table of Contents

David O'Regan: 10331017 david.oregan7@mail.dcu.ie

Introduction

This document is designed to be a reference for any person wishing to implement or any person interested in the architecture of the Dealz Machine Learning Prediction Algorithm, Mining Algorithm or Android Application. This document describes each of the projects core architecture and sub-architecture their associated interfaces, database schemas, and the motivations behind the chosen design. Both high-level and low-level designs are included in this document.

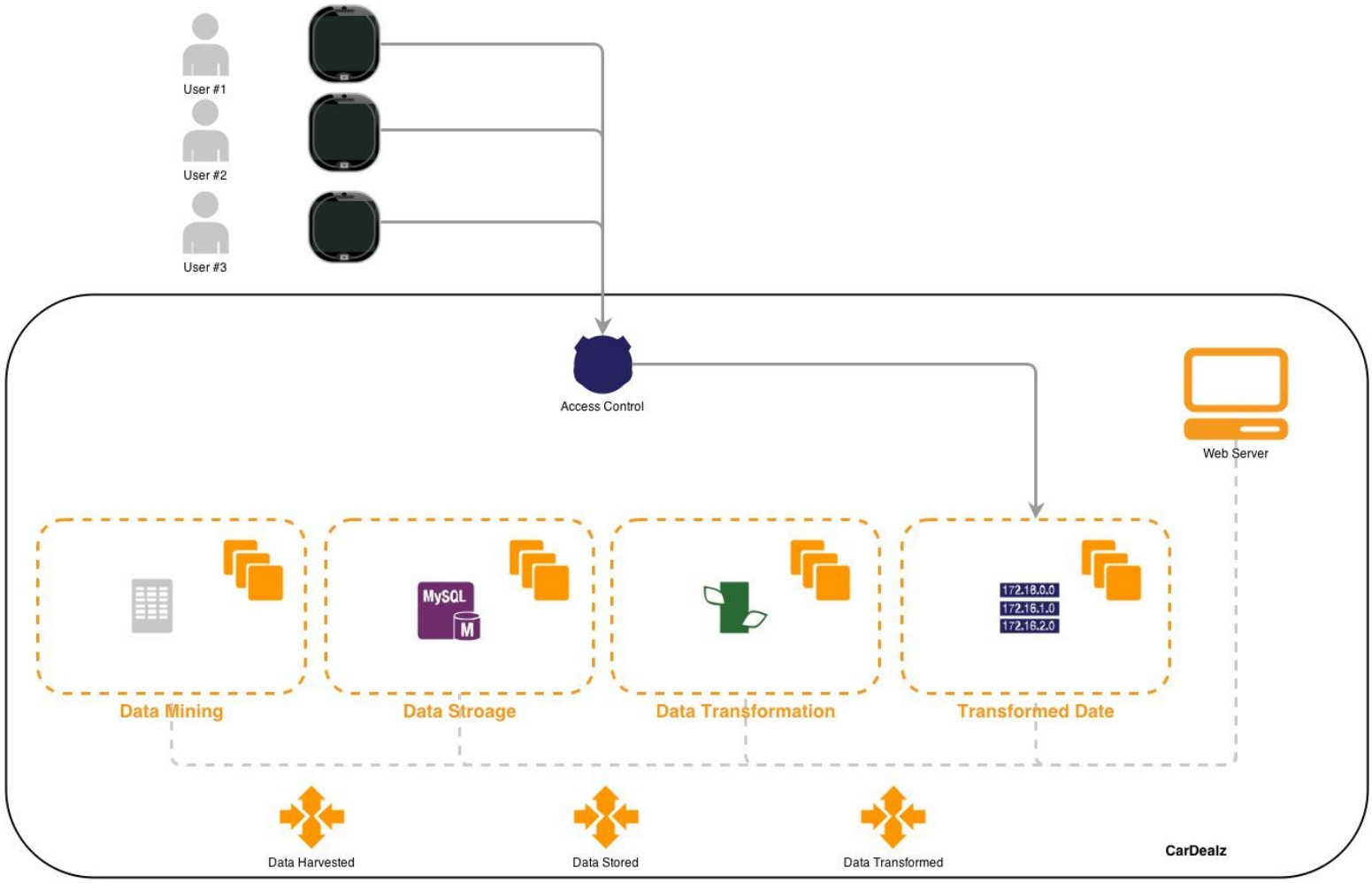
This design document has an accompanying user manual and test document.

This document is specifically aimed at those with a technical profiecnincy. Specifically, those with a good grasp of Python programming, Machine Learning Characteristics, Android development and server/MYSQL management.

This document includes but is not limited to the following information for the CarDealz Prediction System; system overview, design considerations, architectural strategies, system architecture, policies and tactics, and detailed system design.

David O'Regan: 10331017 david.oregan7@mail.dcu.ie

System OverView



David O'Regan: 10331017 david.oregan7@mail.dcu.ie

Design Considerations

This issue describes the issues I encountered when developing my application and how that altered the design process.

Assumptions and Dependencies

This design of the Sports Score system makes several assumptions about software and hardware, and has several software dependencies. All environmental requirements of both the server and Android applications can be found in the CarDealz Requirements 3.1.

Both the server and Android applications make the following assumptions about their environmental environments;

- The system can be described by the environmental requirements associated to this document.
- The system the application is executing on will have the required resources available as necessary. This entails sufficient memory and permanent storage space, an adequate CPU for the necessary application, and a TCP/IP network connection.

The Android application makes the following assumptions about its operation environment;

- The Android machine will an Android device with an API level of at least 4.2.1, which will include HoneyComb.
- The Android machine will have wifi or sufficient 3/4G access.

The server application makes the following assumptions about its operation environment;

- The server machine a MYSQL database which will house the needed application data in the schema outlined in the document.
- The server machine will provide an webspace for the secure storage of stored information that our application may access via HTTP requests.
- Preferably the server machine will have at least 100 GB of free space, with an accompanying 2GB RAM minimum.

David O'Regan: 10331017 david.oregan7@mail.dcu.ie

Goals and Guidelines

The major goal of the CarDealz application is to be suitable for an end user that has little to no programming or Machine Learning knowledge. Our target audience will consist of manly young car enthusiast's as opposed to those familiar with the concept of Machine Learning. For this reason, the end user has minimal knowledge and interaction with the backend of the application, they are simply supplied with a front end application which is designed to be useable by anyone,

The second major goal of the application is that the user gets a response in a timely fashion. Intuition tells that a user will lose interest if they have to wait long times for software to respond. This is why the design has minimal data transferred between Android and server. In this design, a minimum set of information is transferred to the server in order to retrieve the necessary information, and the server only returns the requested data that is then formatted into a accessible link on the Android side.

The Last major goal of the application is that the user gets provided a dataset that is accurate. If our final data set has not trained accurately the user will not use the application. This is why the design process centred heavily around getting the Machine Learning algorithm to train as accurately as possible.

The CarDealz server is never intended to be seen by the end user nor interacted with, hence it is managed without the use of a visual manager.

David O'Regan: 10331017 david.oregan7@mail.dcu.ie

Architectural Strategies

The CarDealz design has been divided into four major sub-systems; mining application, Android application, source database, and regression algorithm. The mining application is then separated into four major sub-sections; the spider component, server communications, data pipeline and string cleaning. The Android application is separated into two major sub-sections; the Android front end display component, and the Android/ Database communications.

The mining application's major design considerations include capturing relative data and easy database updates. The mining application was designed to be as flexible as possible within the constraints of working with Xpath detection. For the purpose of this project, the mining application was limited to a election of cars within ww.carzone.ie

The Regression algorithm is designed to support the following major features; establish a database connection, assign string data into Pandas Frame, assign weights to each conditional value and finally perform the regression to determine an average from which we can establish a good or bad "deal".

The Android application is designed to support the following major features; a simple and intuitive graphical user interface (ListView), easy to use list frames, flexible search options, and support of a weblink that can take the user directly to the car they are currently viewing. Given the system's requirement of the Android application, it must be supported on a API of 4.2.1 or greater.

David O'Regan: 10331017 david.oregan7@mail.dcu.ie

Subsystem Architecture

1 - Mining Application

```
#scrap listing page to get content
def parse_listing_page(self, response):

    hxs = HtmlXPathSelector(response)


    #print "CALL BACK" - Testing

    item = response.request.meta['item']
    item['title'] = hxs.select('//*[@id="car-header"]/h1/text()').extract()[0].strip()
    item['link'] = response.url
    item['price'] = hxs.select('//*[@id="advertDetailsPrice"]/text()').extract()[0].strip()
    item['carYear'] = hxs.select('//*[@id="advertDetailsYear"]/text()').extract()[0].strip()
    item['location'] = hxs.select('//*[@id="advertDetailsLocation"]/text()').extract()[0].strip()
    item['mileage'] = hxs.select('//*[@id="advertDetailsMileage"]/text()').extract()[0].strip()
    item['engine'] = hxs.select('//*[@id="advertDetailsEngine"]/text()').extract()[0].strip()
    item['Transmission'] = hxs.select('//*[@id="advertDetailsTransmission"]/text()').extract()[0].strip(
    item['Colour'] = hxs.select('//*[@id="advertDetailsColour"]/text()').extract()[0].strip()
    item['Owners'] = hxs.select('//*[@id="advertDetailsOwners"]/text()').extract()[0].strip()
    item['NCT'] = hxs.select('//*[@id="advertDetailsNCTExpiry"]/text()').extract()[0].strip()
    item['BodyType'] = hxs.select('//*[@id="advertDetailsBodyType"]/text()').extract()[0].strip()


    yield item
```

Figure 1 – Mining Application Scraping

Scraped data is stored into Database.

```
class MySQLStorePipeline(object):
    def __init__(self):
        self.conn = MySQLdb.connect(user='root', passwd='root', db='carzone',port=8889, host='127.0.0.1',
                                    charset="utf8",
                                    use_unicode=True)
        self.cursor = self.conn.cursor()
```

Figure 2 – Mining Application Storing

David O'Regan: 10331017 david.oregan7@mail.dcu.ie

Scrape Data:
We'll first retrieve data from car zone (www.carzone.ie), check for valid Xpath with each item to be scraped. If the path is valid, scrape the data, otherwise log errors and debug.

Format strings being stored:
Put parsed data in the format that we'll discuss in the interface section. Then check if we get the correct data. The reason that we wait until this part to check the data instead of doing that right after we get the data is efficiency. We don't want spending too much time checking data. If the data is correct, then write it to file. Otherwise, log errors.

Store Data
If the data retrieved OK, open a connection to our server DB and then store data in the database for later use. Otherwise, log errors and debug.

The program (mining.py) should be executed right before midnight to ensure the stable format of the web site. So, we'll have more chance getting the correct data.

To run the schedule program, type: python mining.py

2 - VPS DataBase(MYSQL)
The VPS DB has two distinct functions:

1) The Database is implemented to store both our initial unclassified data and both our classified data(Regression).
2) The VPS will allow the Android application display the data which has been classified by our regression algorithm.

3 - Regression Algorithm
The regression component can be broken up into three distinct sub-components; The database connection, the bag of words model, and the actual regression.

David O'Regan: 10331017 david.oregan7@mail.dcu.ie

Database connection Application

```
#Connect to my DB to get data, place data into a pandas Dataframe
db = MySQLdb.connect(host="mysql.raven.com", user="david", passwd="", port=3306,
                     db="david")
sql = """
        select *
            from audi_a4
      """
df = psql.frame_query(sql, db)
# db.close()
```

The database connection sub-component is used to open the connection to the VPS and access any table which will contain the raw data needed. The database table is opened via pythons native MySQLdb method connect, and from their a query is made to parse the raw data into a Pandas frame for manipulation.

Bag Of Words Model

```
#Now use the sklearn DictVectorizor libary to map each colum from the data frame into a numpy array
# Transforms lists of feature-value mappings to vectors.
#
#
dv = DictVectorizer()
dv.fit(df.T.to_dict().values())
```

Text Analysis is a major application field for machine learning algorithms. However the raw data, a sequence of symbols cannot be fed directly to the algorithms themselves as most of them expect numerical feature vectors with a fixed size rather than the raw text documents with variable length. To accomplish this, our raw data is first stored into a pandas data frame, from which the Sklearn DictVectorizer is applied; this method will transform lists of feature-value mappings to vectors. When feature values are strings, this transformer will do a binary one-hot (aka one-of-K) coding: one boolean-valued feature is constructed for each of the possible string values that the feature can take on. This feature makes applying a Regression algorithm possible.

David O'Regan: 10331017 david.oregan7@mail.dcu.ie

Regression Algorithm

```
# Create linear regression object
LR = LinearRegression()

# Train the model using the training sets(DataFrame without title, link or price and then price by itself)
LR = LR.fit(dv.transform(df_no_priceLinkTitle.T.to_dict().values()), df.price)
```

Figure - DictVectoriser Implementation

We specify a Linear Regression object, which initialises as an empty method. This object is passed two data sets for it to train against; our data frame without the asking price and the asking price in a separate dataframe. Using this format, the asking price will not skew the training of the LR and we can then compare our results later to establish if a given add is a worthwhile deal.

```
goodDeal = (doc['price'] < predicted)
```

Figure - Good Deal compare statement

David O'Regan: 10331017 david.oregan7@mail.dcu.ie

4 - Android Component

The user interface will be designed as three separate pieces--the brand listings, actual listings and the final view that will present this information to our user. The distinction between each of the three frames is large but developed to keep as much uniformity as possible within the application.

The dialogs themselves will not be hard-coded into the system. Rather, they will be read and interpreted from a database structure. An end user needs to know nothing about programming to use the front end design we have provided.

The user interface infrastructure is quite complex, requiring the use of fragments in building list views and retro fit for pulling data into view. However the upside of using fragments is, the code should be fairly compact and easy to maintain. The alternative, coding each frame with an activity separately, would greatly expand the code, would most likely duplicate much of the common functionality several times, and would require generating each list view every time a user went back a page, which is a large memory waste.

Included in the design of the infrastructure is the design of a search bar. The user search bar is implemented using a Searchview within one of the fragments and will provide the user with a filter option.

The design document is written to include all functionality that may potentially be implemented during the course of this project. Some of the features, however, will not be implemented unless time allows (see the requirements document). The system should be implemented in such a way that the architecture remains open to these features even if they are not implemented at the current time.

The purpose of the user interface infrastructure is to take data from a table within the VPS and present it to the end user in a easy to understand way. More specifically, it must read information from the database defining the interface to be presented to the human user. It must present the information to the user and accept a search request. When the user has completed a query, the query must be sent to the list and a new list is presented to the user.

David O'Regan: 10331017 david.oregan7@mail.dcu.ie

The following components will be implemented in the user interface infrastructure to achieve its functionality.

1. A database initialisation routine will be implemented via retro fit at the start of the application. This query is needed to not only pull the information we need into the web application, but also make a query to filter the results via the brand requested by a user.

2. The original fragment will provide a list of all possible brands for selection by the user, this will then translate to a second list view of all possible cars within the brand itself. These two list views will be implemented using a common activity with different fragments.

4.3., 4.4., 4.6., 4.7.
A final frame will be implemented by a separate fragment which will provide a layout of a single item selected by the user. This will display all the relative information in textual format on the top of the page, while a web view is implemented to show the actual original add of the car selected. Finally, a link is provided for the user if they wish to follow the add from the application into a actual browser on the android device.

5. A subsystem will be implemented to present the user with search functionality within the list of cars per brand. This search feature will allow a user to filter results based on popular options like colour, location and price.

```
if (TextUtils.isEmpty(newText)) {
    // mAdapter.getFilter().filter("");
    getActivity().getActionBar().setSubtitle("List");
    grid_currentQuery = null;
} else {
    getActivity().getActionBar().setSubtitle("List - Searching for: " + newText);
    grid_currentQuery = newText;


    mAdapter.findByAnything(grid_currentQuery);
```

Figure - SearchView Implementation

David O'Regan: 10331017 david.oregan7@mail.dcu.ie

5 - Mining Breakdown

The mining component will have a structure that supports the following guidelines:

5.1. All Xpath's must be valid in order to harvest correct data. - spider.py

5.2. A table will be created within our DB to store taken information. - pipeline.py
- 5.2.1. Each row will contain a set of values that conform with the DB structure.
- 5.2.2. Each prompt will store a title, link, year, price, mileage, NCT, bodytype, location, engines size and engine type values respectively.

5.3. A pipeline will be created to store values within the DB table. - pipeline.py
- 5.3.1. Each item in the pipeline will correspond to an item field(column) in the database.
- 5.3.2. Each item will contain the text taken from the path in our spider.
- 5.3.3. Each item will contain a logical flag that will indicate whether this item is the correct type of information and if it should be stored.
- 5.3.4. Each item will return a value native to its textual description.

5.4. Strings will be cleaned before DB insertion. - pipeline.py
- 5.4.1. Each item in the pipeline corresponds to an item being scraped from a particular Xpath.
- 5.4.2. Each item taken is a contains string or int format.
- 5.4.3. Each item will have the content cleaned before being inserted into the DB table.
- 5.4.4. For testing and validation, all items are also stored within a cdv file which will act as a visual testing application.

5.5. DB connection will terminate. - pipeline.py & spider.py
- 5.5.1. After each all pages have been scraped, the DB pipeline will finish and the mining program will terminate.

David O'Regan: 10331017 david.oregan7@mail.dcu.ie

5 - Regression Breakdown

The regression component will have a structure that supports the following guidelines:

6.1. A database connection must be established - LR.py

6.2. A pandas dataframe must be created from DB table - LR.py
- 6.2.1. Each row from the DB table is placed into a Pandas Dataframe(Two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns)).
- 6.2.2. Each column will store a separate value; title, link, year, price, mileage, NCT, bodytype, location, engines size and engine type values respectively.

6.3. Dataframe editing takes place - LR.py
- 6.3.1. Each value in colour(over 400 variations) are refined into a few basic colours for ease.
- 6.3.2. Each value in location(26 possible) are refined into the 5 main areas in Ireland.
- 6.3.3. Mileage and price columns will be edited to remove redundant string values for float conversion.
- 6.3.4. Car year is transformed into age for a more accurate training set.

6.4. DataFrame strings are pre-processed for Regression function - LR.py
- 6.4.1. A dictVecoriser object is created
- 6.4.2. Our new, edited pandas dataframe is transformed, assigning feature with multiple values as a unique feature

6.5. Model is trained, accuracy is determined - LR.py
- 6.5.1. LR model is defined, two separate dataframes passed, datagram without price and just price.
- 6.5.2 LR model variance score is calculated based on training fit.

6.6. Predict function is defined and structured - LR.py
- 6.6.1. Processed LR dataframe is passed into predict function, this takes our dataframe and maps it to a bumpy array
- 6.6.2 Then the actual prediction takes place when the LR model is evaluated. Pandas dict is returned with values assigned to features.

David O'Regan: 10331017 david.oregan7@mail.dcu.ie

6.7. Predict function is implemented- LR.py
    6.7.1. Function called, dictVecoriser and LR both passed into function.

6.8. Results are placed into json file - LR.py
    6.8.1. After results are generated, we dump the results to a json file for use later
6.9. Results are placed into csv for testing - LR.py
    6.9.1. The results are then parsed from JSON to CSV for both testing and DB upload
6.10. Results are placed into DB table - LR.py
    6.10.1. Results are uploaded to DB from local CSV readouts

```
predictedPrice = predictFunction(dv=dv, lr=LR)
print " "
with open('audi_a4.json', 'w') as outfile:
    for i in range(200):
        outputPrice = predictedPrice.predict(predictedPrice.transform(df_no_ID.T.to_dict()[i]))
        # print outputPrice[0]
        json.dump(outputPrice, outfile)
        outfile.write('\n')
```


David O'Regan: 10331017 david.oregan7@mail.dcu.ie

7 - **Android Breakdown**

The android component will have a structure that supports the following guidelines:

7.1. 3 separate fragments for item viewing - BrandFragement/Listfragment/ ItemFragement.java

7.2. BrandFragement
- 7.2.1. Each brand needed to cater for is generated within a list view
- 7.2.2. When a user selects a brand, a HTTP query is passed to the webspace for filtering car type

7.3. ListFragement
- 7.3.1. carAdapter.java is accessed to call information into list view.
- 7.3.2. Each item in the list will be a car of the previous brand type selected.
- 7.3.3. Each item in the list will be a selected “good deal” from our DB table.
- 7.3.4. A user will be able to use the search bar to filter the page results.

7.4. ItemFragement
- 7.4.1. The item displayed is the item selected from previous view
- 7.4.2. For each car displayed, the user will be shown the predicted price, the colour and the location of the car.
- 7.4.3. Each item shown must contain a followable link for the user.
- 7.4.4. Each item shown will contain a web view of the add in question for user evaluation.

7.5. Models - cars.java
- 7.5.1. Define a model of car which will contain the information we wish to display to the user per car i.e. car object class

7.6. Services - CarService.java
- 7.5.1. Using retro fit, make a request to the VPS to populate the listfragement with selected brand of car objects

7.7. Adapter - carAdapter.java
- 7.5.1. Car adapter acts as a bridge for our frame views and the actual array of car objects.

7.8. Activity’s - MainActivity, ItemActitviy.java
- 7.8.1. Main activity will handle the brand and list fragment’s.
- 7.8.2 Item activity will handle the single item fragment.

David O’Regan: 10331017 david.oregan7@mail.dcu.ie

Policies and Tactics - Android/Mining/Regression

This design was attempted to be made as modular as possible. This provides flexibility between component developments.

In design, we attempted to partition the development into sections that each individual could create independent of another, and have a clearly defined interface between components.

This design also took the policy of using coding standards such as standard Java/Python variable prefixes and caption. Generally method/property purposes are easily deciphered by their descriptive name.

David O'Regan: 10331017 david.oregan7@mail.dcu.ie

Research

1 - Mining

The project was to revolve around evaluating carzone.ie, harvesting the data and then using that raw data to find trends(averages,means etc).

During my Functional Spec, it was decided that using Scrapy(Python) in order to Scrape my data was the choice made. This choice was made due to the large amount of resources available for making a spider within Scrapy and having it work successfully.

After some deliberation, it was decided that this choice would be kept as it provided the best learning curve and offered the most flexibility for transporting data out to a database table.

2 - Machine Learning

The courseEra machine learning program was chosen for initial research. This course ran for 12 weeks taught by Andrew Ng from Stanford. The course, done in conjunction with my other modules allowed me to develop a basic idea of how machine learning works and how it may be applied in a programming sense.

Python seemed to be universally accepted as an excellent language for machine learning, second only to R and Mathlab. Considering the previous Python experience it was decided it would be the base language over R/Mathlab.

To begin with, pybrain was considered as a library for ML. PyBrain is a modular Machine Learning Library for Python. Its goal is to offer flexible, easy-to-use yet still powerful algorithms for Machine Learning Tasks and a variety of predefined environments to test and compare your algorithms.

The hope was, that this library would allow the structuring of a proper classification algorithm on the dataset and produce a reasonable regression model.

Pybrains solution was a feed forward netowrk, which translates to a neural network. A long time was spent trying to implement this structure of NN for the ML program but in the end, it simply would not accept the multitude of data or classification was working with.

Finally, after trading emails with Andrew Ng from Stanford, it was decided sklearn would be the Library of choice for the ML algorithm.

David O'Regan: 10331017 david.oregan7@mail.dcu.ie

3 - Android

Android was a completely new concept for the development as a whole in this project which meant a steep and difficult learning curve. It was however highly rewarding.

Issues & Solutions

1 - Mining

The main issues around mining stemmed from three areas:

1) Defining the correct Xpath

Defining the correct Xpath for certain functions became a rather difficult issue when trying to develop the scraper. Given the setup of the car zone webpages, items like "follow to next page" were defined in misleading Xpaths.

Solution:

Trail and error quickly became a redundant idea as there was no uniform pattern between items within a page. To work around this, firebug was employed to automatically extract the Xpath for most wanted items. Some however still needed to be evaluated by hand.

2) Scraping enough data

For the dataset to train effectively, it need a large corpus of data to pull from. Carzone simply did not have enough listings per model to make for a extremely impressive dataset.

Solution:

The solution would have involved scraping multiple car websites for the same model and deleting redundant duplicates. This however did not happen for various reasons.

3) Getting relevant information from the site

The scraper needed to pull the relevant information which would let the machine learning algorithm train in the most effective way possible. This meant the scraper needed to access each individual add on the page, scrape the information and then return to the main list to continue repeating the process.

Solution:

David O'Regan: 10331017 david.oregan7@mail.dcu.ie

2 - Machine Learning

The main issues around ML stemmed from three areas:

1) Transforming data into ML friendly input

This was a large stumbling block within the ML side of the application for almost 3 weeks. There had been a large selection of methods tried to get around the issue and effectively transform the string data for manipulation.

Solution:

A combination of Pandas Dataframes and Sklearn DictVectorizer method

2) Training LR effectively

After the data had been effectively transformed and the model was trained. It began apparent that accuracy was an issue and needed to be worked on. The accuracy error stemmed from simple exceptions within the same set(most extreme on the scale) and the overall lack of data to learn from.

Solution:

Regex was implemented to account for large variations within the dataset, the dataframe was manipulated to reduce the volume of choice within columns and a log of accuracy was kept.

3) Upgrading from LR

LR was considered too simplistic an approach for the project and it was encouraged to upgrade to a more complicated algorithm.

Solution:

After some time, a SVM was implemented but due to the lack of data was actually less accurate than the LR. After some testing, it became apparent that the issue stemmed from overfitting; “Overfitting generally occurs when a model is excessively complex, such as having too many parameters relative to the number of observations. A model which has been overfit will generally have poor predictive performance, as it can exaggerate minor fluctuations in the data.”.

The solution was to use the LR as our base algorithm.

David O’Regan: 10331017 david.oregan7@mail.dcu.ie

3 - Android

The main issues around UI stemmed from three areas:
1) Successfully pulling data from webspace into application

Solution:

Implement Retro Fit to post a HTTP request to my servers webspace and the implementation i explained here: http://blogs.computing.dcu.ie/wordpress/oregand7/2014/05/16/retrofit-for-my-Android-application/

2) Search Bar Functionality

Solution:

Implement a custom filter algorithm to limit the displayed array of results based on a user input.

3) Allowing for the user to return to a previous list

Solution:

Store a previous array in a state of memory and reload that array if the user selected the backspace with the filter.

Testing

There is a testing booklet provided with this specification. It contains all relevant test cases and UNIT testing descriptions.

David O'Regan: 10331017 david.oregan7@mail.dcu.ie

Results

1 - Mining

The results were successfully mined and stored within the Non-relational DB that had been created to store the harvested data.

2 - Machine Learning

The model was successfully fitted and trained using the data taken from carzone.ie

3 - Android

The application successfully allows the user to visually view those car zone items that are considered “good deals”.

Acronyms and Abbreviations

ML	Machine Learning
LR	Linear Regression
Mining	Data Scraping
DB	Database

David O’Regan: 10331017 david.oregan7@mail.dcu.ie