COMP1005 Week 12 Cheat Sheet

Lisa Luff

11/7/2020

Contents

Database Access

- Specialised computer system built to store and retrieve data
- Relational databases were developed to minimise repetition to use less space and reduce inconsistencies (like functions)
- They use the relationships between data items to split the information into tables
- The most common way to access a database is using SQL
  - Python has a standard implementation for working with SQL databases - the Database Application Programming Interface (API)
- The API defines how to:
  - Import the API module
  - Acquire a connection with the database
  - Issue SQL statements and stored procedures
  - Close the connection
- mySQL is a common type of database and has a module MySQLdb
  - MySQLdb is an interface for connecting to a MySQL database server from Python
  - It implements the Python Database API v2.0 and is built on top of the MySQL C API
- Relational databases are made up of tables designed to minimise duplication
  - Now new options are available that use flat files and distributed data for fast performance ahead of storage efficiency
  - No SQL databases are used in big data applications, developed by Google, Yahoo, Facebook and Amazon

MySQLdb

- Implementation of MySQLdb:
  - Initiate -
    import MySQLdb
    db = MySQLdb.connect("*localhost*", "*user*", "*file*", "*database*")
  - Prepare cursor object
    cursor = *db*.cursor()
  - Execute SQL query
    cursor.execute("*SELECT VERSION()*")
  - Fetch a single row
    data = *cursor*.fetchone()
  - Disconnect from server
    db.close()
  - Drop table if it already exists
    cursor.execute("*DROP TABLE IF EXISTS EMPLOYEE*")
  - Create table
    table = """*CREATE TABLE (INSTRUCTIONS)*"""
    cursor.execute(*table*)
  - Insert a table
    table = "*INSERT TABLE (COLUMNS) \ VALUES ('%ROWS') % \ ('ROWVALUES')*"
    try:
    cursor.execute(*table*)
    db.commit()
    except:
    db.rollback()
  - Extract data
    items = *cursor*.fetchall()
    for row in *items*:
    a = row[*n*]

Web Scraping

- Web scraping is a way of extracting information from websites
- There are many packages available for web scraping
- You should always look for an API to access web data before going too far with scraping
  - API's are provided by the organisations for easier access to the data underlying the site
  - Packages often change format, then your scraper will break
- HTML (web pages) have various tages described below:
  - <!DOCTYPE html> - Start with a type declaration
  - Document contained between
  - Visible part of the document is between
  - Heading are defined with tags
  - Paragraphs are defined with the
    tag
  - Links are defined with
  - Tables are defined with
    , row as
    and rows are divided into data as
  - Lists start with
    (unordered) and
    (ordered), each item of the list starts with
- Accessing and parsing a web page:
  import urllib.request
  - Using BeautifulSoup as API
    from bs4 import BeautifulSoup
  - Specify the URL
    site = "*link*"
  - Collect html from the URL
    webpage = urllib.request.urlopen(*site*)
    code = BeautifulSoup(*webpage*, 'html.parser')
  - Make code readable
    print(soup.prettify())
  - Extract a table *table* = *code*.find('table', class_='*table type*')
    rows = *table*.find_all("tr")
    for row in *rows*:
    cells = row.find_all('td')
    a.append(cells[*n*].get_text())

Internet of Things

- Incoming data:
  - We can set up Things, and help them to communicate
  - Need to provide storage to bring them to rest, ready for analysis
  - Types of data
    - Sensor data - Time series, usually numeric, orientation, movement, temperature, light, etc.
    - Video/images
    - Locational
    - Count
- Sensor data:
  - Collected from channels and provided in a raw state
  - Individual sensors and devices don't know their place in the real world, although they will come with some identification
  - Requires encoding, decoding, conversion, stripping and tagging
  - Might also combine with related data
  - We can use statistical tools to look for problems and fix them

Cleaning Data

- Things live in the real world, which is messy and inconsistent
- Data might have been designed for other purposes
- Raw data must be cleaned to be technically correct and consistent
- Provides confidence in the results of our analysis
- Cleaning should be systematic and documented allowing for reproducibility and automation
- Basic data cleaning:
  import pandas as pd
  import numpy as np
  def clean(data):
  data = *data*.replace(*item*, np.nan) # if need to make some values nan
  data = data.dropna (axis = 0, how = 'any') # 0 is for rows, 1 for columns
  data['*Date Time*'] = pd.to_datetime(data['*Date Time*'])
  return *data*
  - Combining data sources
    sensor = rd.read_csv('*filename*')
    sensor = clean(data = *sensor*)
    other = pd.read_csv(*otherfile*)
    other = clean(data = *otherfile*)
    start = '*datetime string*'
    end = '*datetime string*'
    sensor = timeframe(*start*, *end*, *sensor*)
    other = timeframe(*start*, *end*, *sensor*)
    ax = *sensor*.plot(*x*, *y*)
    other.plot(*x*, *y*) # plots them together

Cloud-Based DASHBOARD Services

- Many free + subscriber options availabel to log and display data
- Register device with Cloud service
- Select display options
- Add alarms and notifications
- Publish from device to service or Control device from service
- Examples:
  - Thingspeak
  - Freeboard
  - IFTTT
  - Thingsboard
  - InitialState
- Cloud-based storage allows collation of large amount of data
- The dashboards are effective and impressive
  - Lots of real-time data, ability to merge datasets and options for alerts and alarms
  - However, we often don't have a specific threshold we can set, the values to create the state we're interested in may not be obvious
  - Need to explore the data to find patterns that predict future behaviour
  - Machine learning can help

InitialState

- Code to push to InitialState
  from ISStreamer.Streamer import Streamer
  ACCESS_KEY = '*key*'
  BUCKET_KEY = '*key*'
  BUCKET_NAME = '*name*'
  streamer = Streamer(bucket_name=*BUCKET_NAME*, bucket_key=*BUCKET_KEY*, access_key=*ACCESS_KEY*) # Creater streamer instance
  streamer.log("*label*", *info*) # Send some data
  streamer.flush() # Close the stream

Data Management

- Although you have access to "unlimited" data storage (at a price), need to plan and understand the data
- Document sources and provenance
- To reduce costs (and confusion) may not need to:
  - Store/keep everything
  - Keep data at the source resolution (downsample)
  - Keep each intermediate form of the data
  - Keep all fields (after identification and tagging)
- One way to deal with "Big Data" is to make it less big

Machine Learning

- About designing algorithms for creating models that can take combinations of inputs and assign an output class/decision
- The models can automatically update themselves, training/retraining and updating training/retraining and updating training sets to refine or discover new rules
- Core algorithms include clustering and supervised classification, rule systems, and scoring techniques
- The ability to work backwards from an end goal, without needing to specify the variables needed to acheive that goal, is one of the most valuable aspects of machine learning technology
- Applications:
  - Predictive maintenance - Laying audio and image data to improve preventative maintenance
  - AI-driven logistics optimisation - 15% reduction in fuel costs by monitoring and coaching driver habits
  - Customer service management and personalistion - Recognise emotion in customers voices and send them throguh to real people
  - Object recognition algorithms for photo tagging on social media
  - Facilitate image analysis for imaging applications like microscopy using deep learning
  - Object detection to create a smarter checkout experience in retail stores
- Example ML workflow:
  - Identification of the problem
  - Evaluation of the data size and type
  - Data transformation and exploration
  - Model identification, training, testing
  - Model deployment
  - Data visualisation
- Part of an ecosystem to deliver understanding and action
- Tools/frameworks:
  - Tensorflow - A machine learning library originally developed by the Google Brain team for internal use, now available to everyone
  - Scikit-learn - Machine learning development in Python. Provides a library for the Python programming language
  - Keras - An API for neural networks. Helps doing quick research and is written in Python
  - Amazon Machine Learning (AML) - A robust, cloud-based machine learning and artificial intelligence software which integrates data from multiple sources
  - Microsoft Azure Machine Learning Studio - a collaborative, drag and drop tool that can be used to build, test, and deploy predictive analytics solutions on data. Publishes models as web services to be consumed by custom apps or BI tools

Language Comparator

- There is a program to convert between languages by Dave Cooper

Object Detection

- Very popular application of artificial intelligence
- Computer vision techniques can be ised to detect objects in images and video
- Given an input image/stream, we will generate:
  - A list of bounding boxes (x, y) coordinates for each object
  - A class label for each bounding box
  - A probability/confidence score for each box/label
- Deep learning object detection:
  - They train our framework to perform object detection, modifying both the weights of the new layers/modules and base network
- Example:
  import argparse
  import cv2
  - Parse the arguments
    ap = argparse.ArgumentParser()
    ap.add_argument("-i", "–image", required=TRUE, help="path to the input image")
    ap.add_argument("-c", "–cascade", default="haarcascade_frontalcatface.xml", help="path to cat detector harr cascade")
    args = vars(ap.parse_args())
  - Load input image (in greyscale)
    image = cv2.imread(args["*image*"])
  - Load detector Haar cascade, then detect input image
    detector = cv2.CascadeClassifier(args["cascade"])
    rects = *detector*.detectMultiScale(*colourscale*, scaleFactor=*n*, minNeighbours=*m*, minSize=(*a*, *b*))
  - Loop over images and draw rectangles
    for (i, (x, y, w, h)) in enumerate(*rects*):
    cv2.rectangle(image, (x, y), (x + w, y + h), (0, 0, 255), 2)
    cv2.putText(*image*, "Cat #{}".formate(i + 1), (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.55, (0, 0, 255), 2)
  - Show detected faces
    cv2.imshow(*image*)
    cv2.waitKey(0)
  - Will print the image with rectangles over the cat faces