Data Science: Techniques and Tools (DRAFT)

Mark A. Austin

University of Maryland

austin@umd.edu
ENCE 688P, Spring Semester 2022

December 13, 2022

Overview

Getting Started

Definition of Data Science

Various Sources (Google, ScienceDirect):

- .....
- .....

Basic Data Science:

- ....
- ....
- ....
- ....

Framing the Opportunity

We seek:

- Data-driven approaches to measurement of performance in the building environment and identification of trends and patterns in behavior.
- Solutions that account for unique physical, economic, social and cultural characteristics of individual cities.

Sources of Complication:

- Multiple domains; multiple types of data and information.
- Network structures that are spatial and interwoven.
- Behaviors that are distributed and concurrent.
- Many interdependencies among coupled urban subsystems.

Data Science Techniques

(Useful things to know when building examples)

Related Data Science

Topics

- One-Hot Encoding Techniques
- ETL (Extract-Transform-Load) Processes
- Iterative Strategies of Learning
- Data Organization: Sample, Batch size, and Epochs

One-Hot Encoding

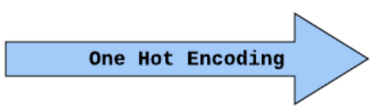
One-Hot Encoding

One-Hot Encoding

- One hot encoding is one method of converting data to prepare it for an algorithm and get a better prediction.
- Each categorical value is converted into a new categorical column and assign a binary value of 1 or 0 to those columns.
- Each integer value is represented as a binary vector.

Simple Example (Source: datascience.com):

id	color
1	red
2	blue
3	green
4	blue



id	color_red	color_blue	color_green
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0

One-Hot Encoding

Example 1: One-Hot Encoding in Python

```
1  # ======================================================
2  # TestEncoder01.py: Manual one hot encoding with numpy
3  # ======================================================
4
5  from numpy import argmax
6
7  print("TestEncoder01.py ... ")
8  print("======================================================")
9
10 # define input string
11
12 data = 'hello world'
13 print("--- Input string: %s" %(data))
14
15 # define universe of possible input values
16
17 alphabet = 'abcdefghijklmnopqrstuvwxyz '
18 print("--- Alphabet of input values: %s" %(alphabet))
19
20 # define a mapping of chars to integers
21
22 char_to_int = dict((c, i) for i, c in enumerate(alphabet))
23 int_to_char = dict((i, c) for i, c in enumerate(alphabet))
24
25 # integer encode input data
26
27 integer_encoded = [char_to_int[char] for char in data]
28 print("--- Integer encoded data: %s" %(integer_encoded))
```

One-Hot Encoding

Example 1: continued ...

```
29
30  # one hot encode
31
32  print("--- One hot encode ...")
33
34  onehot_encoded = list()
35  for value in integer_encoded:
36          letter = [0 for _ in range(len(alphabet))]
37          letter[value] = 1
38          onehot_encoded.append(letter)
39  print(onehot_encoded)
40
41  # invert encoding
42
43  print("--- Invert encoding ...")
44
45  print("--- Encoding[ 0] : %s" %( int_to_char[argmax(onehot_encoded[0])] ))
46  print("--- Encoding[ 1] : %s" %( int_to_char[argmax(onehot_encoded[1])] ))
47
48  .... lines of code deleted ...
49
50  print("--- Encoding[ 8] : %s" %( int_to_char[argmax(onehot_encoded[8])] ))
51  print("--- Encoding[ 9] : %s" %( int_to_char[argmax(onehot_encoded[9])] ))
52  print("--- Encoding[10] : %s" %( int_to_char[argmax(onehot_encoded[10])] ))
53
54  print("=======================================================")
55  print("Finished !! ")
```

One-Hot Encoding

Output: ...

```
TestEncoder01.py ...
==================================================
--- Input string: hello world
--- Alphabet of input values: abcdefghijklmnopqrstuvwxyz
--- Integer encoded data: [7, 4, 11, 11, 14, 26, 22, 14, 17, 11, 3]
--- One hot encode ...
[[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 0, 0],
 [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, ... 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, ... 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, ... 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 0, 1],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, ... 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, ... 0, 0, 0],
 [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 0, 0]]
```

One-Hot Encoding

Example 2: One-Hot Encoding in Python

```
1  # ======================================================
2  # TestEncoder02.py: One hot encoding with sklearn
3  # ======================================================
4
5  from numpy import array
6  from numpy import argmax
7  from sklearn.preprocessing import LabelEncoder
8  from sklearn.preprocessing import OneHotEncoder
9
10 print("TestEncoder02.py ... ")
11 print("======================================================")
12
13 # Input string of temperature levels ...
14
15 data = ['freeze', 'cold', 'warm', 'cold', 'hot', 'burn',
16           'warm', 'burn', 'cold', 'warm', 'hot']
17 values = array(data)
18
19 print("--- Input values: %s" %(values))
20
21 # integer encode
22
23 print("--- Integer encode ...")
24
25 label_encoder   = LabelEncoder()
26 integer_encoded = label_encoder.fit_transform(values)
```

One-Hot Encoding

Example 2: One-Hot Encoding in Python

```
27
28  print(integer_encoded)
29
30  # binary encode
31
32  print("--- Binary encode ...")
33
34  onehot_encoder  = OneHotEncoder(sparse=False)
35  integer_encoded = integer_encoded.reshape(len(integer_encoded), 1)
36  onehot_encoded  = onehot_encoder.fit_transform(integer_encoded)
37
38  print(onehot_encoded)
39
40  print("--- Invert encoding ...")
41
42  print("--- Encoding[ 0] : %s"
43        %( label_encoder.inverse_transform([argmax(onehot_encoded[0, :])])))
44  print("--- Encoding[ 1] : %s"
45        %( label_encoder.inverse_transform([argmax(onehot_encoded[1, :])])))
46  print("--- Encoding[ 2] : %s"
47        %( label_encoder.inverse_transform([argmax(onehot_encoded[2, :])])))
48  print("--- Encoding[ 3] : %s"
49        %( label_encoder.inverse_transform([argmax(onehot_encoded[3, :])])))
50
51  print("=======================================================")
52  print("Finished !! ")
```

One-Hot Encoding

Output: ...

```
TestEncoder02.py ...
================================================
--- Input values: ['freeze' 'cold' 'warm' 'cold' 'hot' 'burn'
                   'warm' 'burn' 'cold' 'warm' 'hot']
--- Integer encode ...
[2 1 4 1 3 0 4 0 1 4 3]
--- Binary encode ...
[[0. 0. 1. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 1.]
 [0. 1. 0. 0. 0.]
 [0. 0. 0. 1. 0.]
 [1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 1.]
 [0. 0. 0. 1. 0.]]
```

One-Hot Encoding

Output: continued ...

```
--- Invert encoding ...
--- Encoding[ 0] : ['freeze']
--- Encoding[ 1] : ['cold']
--- Encoding[ 2] : ['warm']
--- Encoding[ 3] : ['cold']
================================================
Finished !!
```

Source Code: See: python-code.d/encoder/

One-Hot Encoding

Advantages

- Computational elements are binary (instead of ordinal) and sit in an orthogonal vector space.

Disadvantages

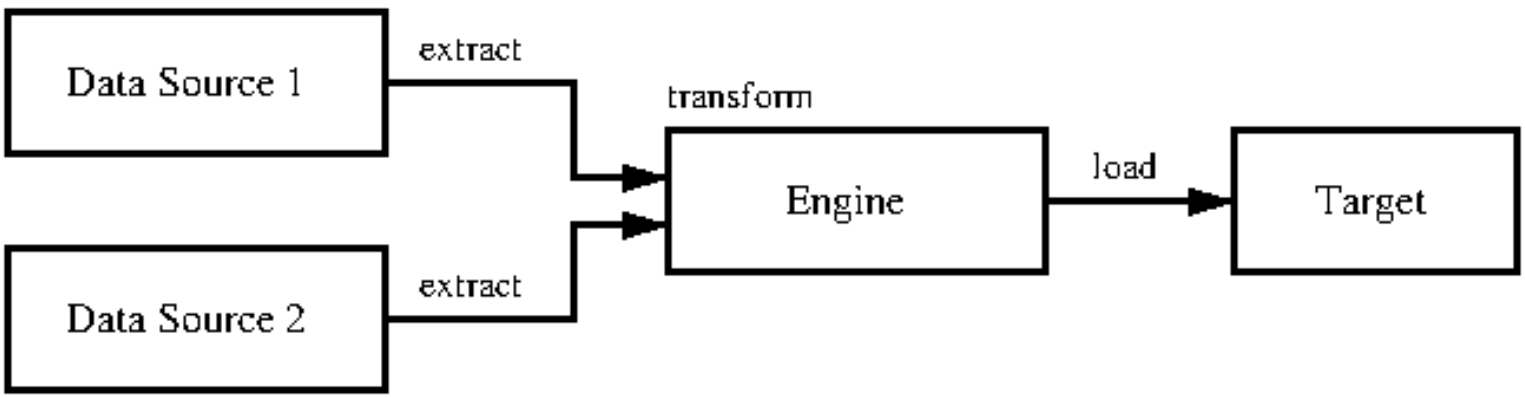
- Some decision tree based methods can work directly with labeled entries – no need for one-hot encoding.
- For high cardinality the vector space can quickly blow up, leading to sparse representations and the curse of dimensionality.
- One solution approach: apply one-hot encoding, then reduce problem space with PCA.

Extract-Transform-Load Processes

Extract-Transform-Load Processes

ETL Processes

- ETL stands for extract, transform, load.
- Traditional ETL extracts data from excel tables, csv, XML, JSON files, etc, and transforms it for storage in centralized databases.



- Emerging ETL extracts data from sensors, mobile Apps, etc, and transforms it for storage in cloud computing.

Extract-Transform-Load Processes

Benefits of ETL

- **Information Clarity.** Data is cleaned and joined across sources before it is saved in a database.
- **Information Completeness.** A well-defined ETL includes all of the data sources relevant to decision making operations.
- **Information Quality.** ETL processes validate data at extraction or correct/discard data at transformation.
- **Information Velocity.** ETL processes can be triggered when new data arrives.

Challenges of ETL

- Traditional targets (databases) are being replaced by cloud computing.

ETL with Pandas

ETL in Pandas

Data Transformation Operations

- Read data in a variety or formats (e.g., csv, text).
- Find and remove duplicate values.
- Remove unnecessary columns; rename columns.
- Filter data to keep only specific values (e.g., "MD" or "VA").
- Conditionally replace invalid values with new values computed by an external function.
- Convert categorical data into integers and one-hot encodiings.
- Extract lower-level detail (e.g., day, hr, min) from string.
- Transfer data to dataset with a function/mapping.

Source Code: See: python-code/pandas/

ETL in Pandas

Example 1: ....

....

Output:

....

ETL in Pandas

Example 2: ....

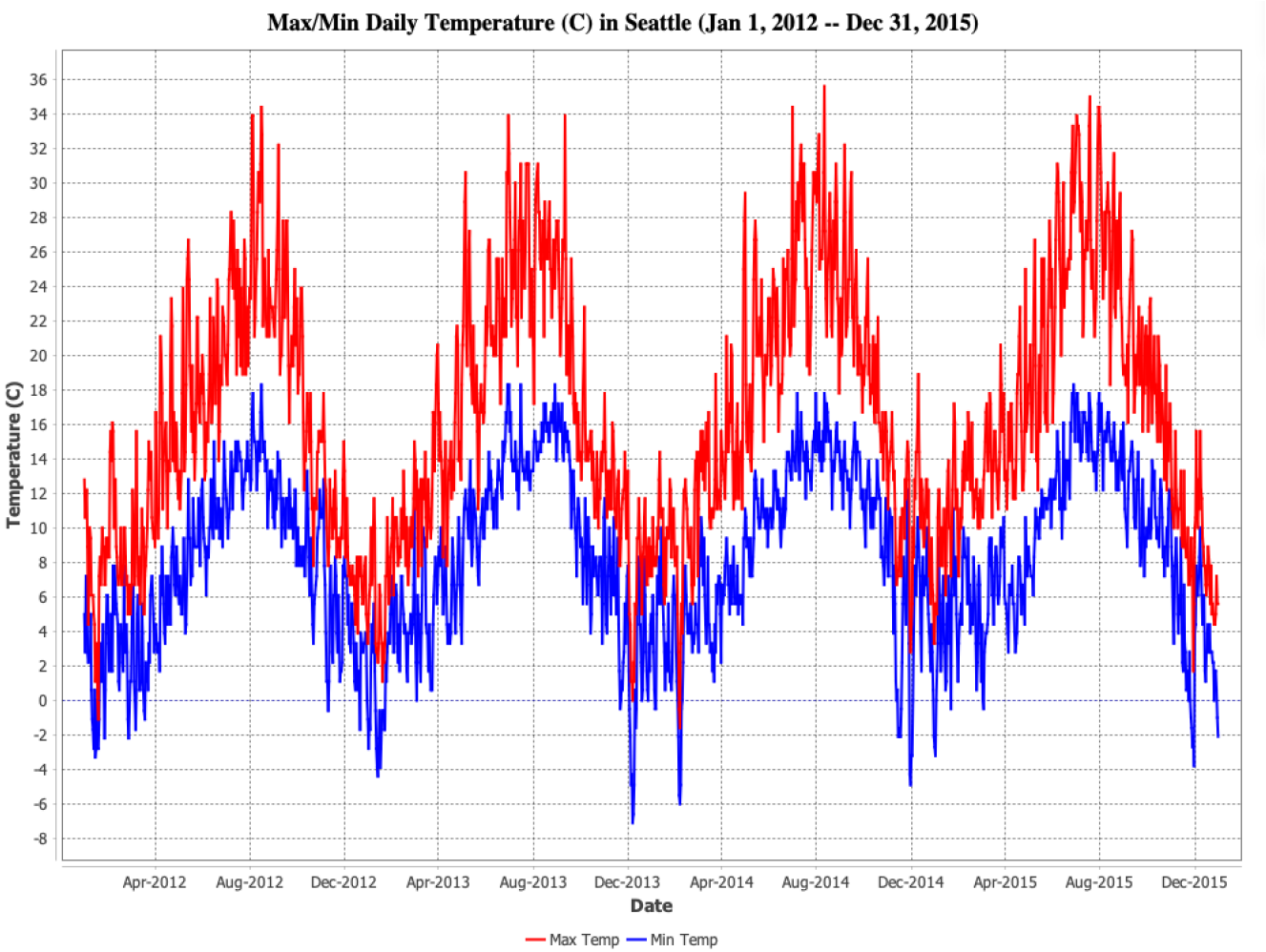
....

Output:

....

ETL in Pandas

Example 3: Process min/max daily temperatures in Seattle



ETL in Pandas

Weather Data in CSV format

Daily Weather Measurements: Jan 1, 2012 through Dec. 31, 2015

```
Date,Precipitation,TempMax,TempMin,Wind,Weather
2012-01-01,0.0,12.8,5.0,4.7,drizzle
2012-01-02,10.9,10.6,2.8,4.5,rain
2012-01-03,0.8,11.7,7.2,2.3,rain

.... data removed ...

2015-12-27,8.6,4.4,1.7,2.9,rain
2015-12-28,1.5,5.0,1.7,1.3,rain
2015-12-29,0.0,7.2,0.6,2.6,fog
2015-12-30,0.0,5.6,-1.0,3.4,sun
2015-12-31,0.0,5.6,-2.1,3.5,sun
```

ETL in Pandas

Solution Procedure:

- Load data.
- Extract day, month and year from date.
- ...
- ...
- ...
- ...

ETL in Pandas

Pandas: Load data

....

Pandas: Extract day, month and year from date.

....

ETL with

Apache DataVec

Working with Apache DataVec

Apache DataVec

- Apache DataVec is an open source Java library for machine learning ETL.
- ETL operations transform raw data into usable vector formats that can be fed to machine learning algorithms.
- Apache DataVec has builtin transformation tools to convert and normalize data.

Data Schema

- A data schema is a high-level blueprint for how a data source (or database) is organized.
- Can think of the schama as being a logical model for how a data model (or database) will be configured.

Working with Apache DataVec

Data Transformation Operations:

- Read data in a variety or formats (e.g., csv, text, image).
- Remove unnecessary columns; rename columns.
- Filter data to keep only examples having specific values (e.g., "NZ" or "USA").
- Conditionally replace invalid values with new values computed by an external function.
- Convert categorical data into integers and one-hot encodiings.
- Parsing a data string and extracting lower-level detail (day, hr, min).

Source Code: See: java-code-ml-dl4j2021/src/datavec/

Working with Apache DataVec

Example 1: Consider the abbreviated data file:

Working with Apache DataVec

Data Schema:

Working with Apache DataVec

Setup Data Transformation Process:

Working with Apache DataVec

Execute Data Transformation Process:

Working with Apache DataVec

Transformed Data Format:

Data Organization

Data Organization

Sample

A sample is simply a single line of data.

Batch

The batch size is a hyperparameter of gradient descent that controls the number of training samples to work through before the internal parameters are adjusted to update the model.

Epoch

An epoch is a hyperparameter of gradient descent that represents a complete pass through the entire training dataset.

Data Organization

Epochs and Batches

- Epochs are comprised of one or more batches.
- The number of epochs can be large, hundreds or even thousands.
- Some learning algorithms require that the batch size and number of epochs be specified upfront.

Common Batch Sizes in RNN

- Powers of two ...
- 32, 62, 128.

Epoch vs Batch Size

References

- Apache DataVec. Library for machine learning ETL (Extract, Transform, Load) Operations. See: https://github.com/deeplearning4j/DataVec.
- Nielsen A., Practical Time Series Analysis: Prediction with Statistics and Machine Learning, OReilly, 2020.