Representation: Feature Engineering

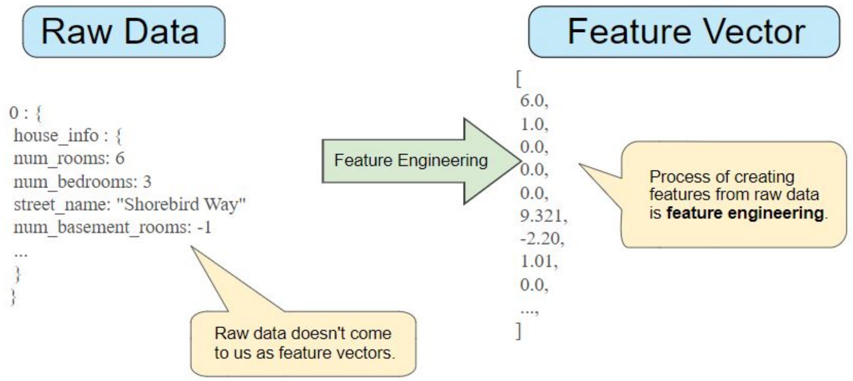
1.

Mapping features

Mapping Raw Data to Features

The left side of the following Figure illustrates raw data from an input data source; the right side illustrates a feature vector, which is the set of floating-point values comprising the examples in your data set. Feature engineering means transforming raw data into a feature vector. Expect to spend significant time doing feature engineering.

Many machine learning models must represent the features as real-numbered vectors since the feature values must be multiplied by the model weights.

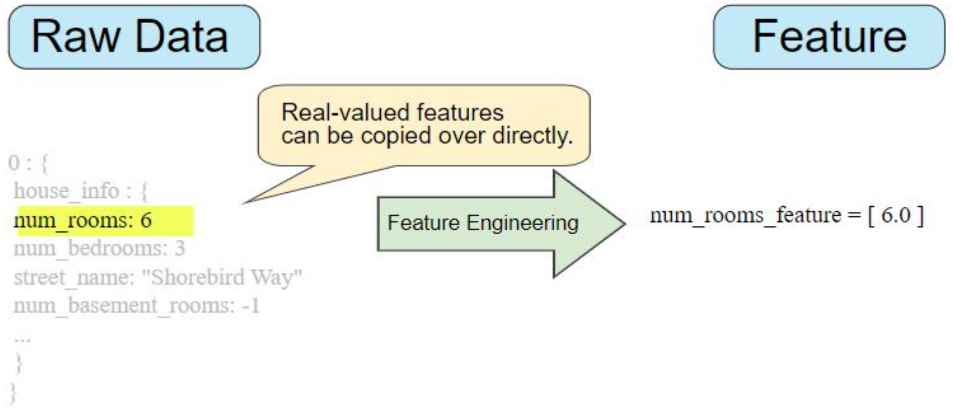

Raw Data
Feature Vector
Feature Engineering
Raw data doesn't come to us as feature vectors.
Process of creating features from raw data is **feature engineering**.

```
0 : {
 house_info : {
 num_rooms: 6
 num_bedrooms: 3
 street_name: "Shorebird Way"
 num_basement_rooms: -1
 ...
 }
}
```

```
[
 6.0,
 1.0,
 0.0,
 0.0,
 0.0,
 9.321,
 -2.20,
 1.01,
 0.0,
 ...,
]
```

Mapping numeric values

Integer and floating-point data don't need a special encoding because they can be multiplied by a numeric weight. As suggested, converting the raw integer value 6 to the feature value 6.0 is trivial:



```
0 : {
 house_info : {
 num_rooms: 6
 num_bedrooms: 3
 street_name: "Shorebird Way"
 num_basement_rooms: -1
 ...
 }
}
```

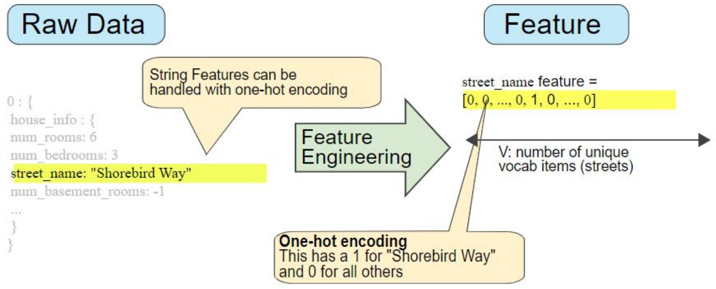
```
num_rooms_feature = [ 6.0 ]
```

Mapping categorical values

Categorical features have a discrete set of possible values. For example, there might be a feature called street_name with options that include:

{'Charleston Road', 'North Shoreline Boulevard', 'Shorebird Way', 'Rengstorff Avenue'}

Since models cannot multiply strings by the learned weights, we use feature engineering to convert strings to numeric values. This representation is called a **one-hot encoding**.


Raw Data
Feature
String Features can be handled with one-hot encoding
0 : {
house_info : {
num_rooms: 6
num_bedrooms: 3
street_name: "Shorebird Way"
num_basement_rooms: -1
...
}
}
Feature Engineering
street_name feature =
[0, 0, ..., 0, 1, 0, ..., 0]
V: number of unique vocab items (streets)
One-hot encoding
This has a 1 for "Shorebird Way" and 0 for all others

Representation: Qualities of Good Features

- Good feature values should appear more than 5 or so times in a data set.
- If a feature's value appears only once or very rarely, the model can't make predictions based on that feature.
- Each feature should have a clear and obvious meaning to anyone on the project. (house_age_years: 27 vs house_age: 851472000)
- In some cases, noisy data (rather than bad engineering choices) causes unclear values. (user_age_years: 277)
- Good floating-point features don't contain peculiar out-of-range discontinuities or "magic" values.

  quality_rating: 0.82

  quality_rating: 0.37

  quality_rating: -1

- To explicitly mark magic values, create a Boolean feature that indicates whether or not a quality_rating was supplied. Give this Boolean feature a name like is_quality_rating_defined. In the original feature, replace the magic values as follows:

  For variables that take a finite set of values (discrete variables), add a new value to the set and use it to signify that the feature value is missing.

2.
Cleaning data

Scaling feature values

Scaling means converting floating-point feature values from their natural range (for example, 100 to 900) into a standard range (for example, 0 to 1 or -1 to +1). If a feature set consists of only a single feature, then scaling provides little to no practical benefit. If, however, a feature set consists of multiple features, then feature scaling provides the following benefits:

- Helps gradient descent converge more quickly.
- Helps avoid the "NaN trap," in which one number in the model becomes a NaN (e.g., when a value exceeds the floating-point precision limit during training), and—due to math operations—every other number in the model also eventually becomes a NaN.
- Helps the model learn appropriate weights for each feature. Without feature scaling, the model will pay too much attention to the features having a wider range.

You don't have to give every floating-point feature exactly the same scale. Nothing terrible will happen if Feature A is scaled from -1 to +1 while Feature B is scaled from -3 to +3. However, your model will react poorly if Feature B is scaled from 5000 to 100000.

Scaling feature values

One obvious way to scale numerical data is to linearly map [min value, max value] to a small scale, such as [-1, +1].

Another popular scaling tactic is to calculate the Z score of each value. The Z score relates the number of standard deviations away from the mean. In other words:

scaledcalue=(value-mean)/stddev

For example, given:

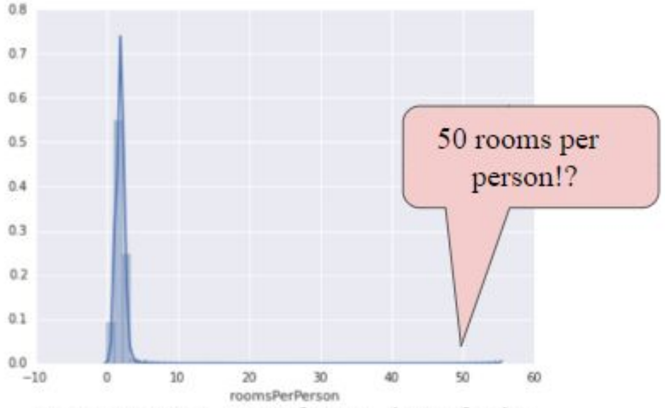
```
mean = 100
standard deviation = 20
original value = 130
```

then:

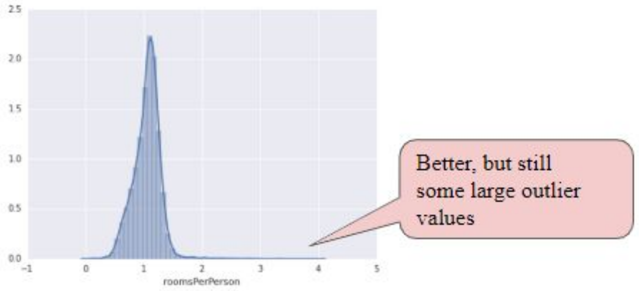
```
  scaled_value = (130 - 100) / 20
  scaled_value = 1.5
```

Scaling with Z scores means that most scaled values will be between -3 and +3, but a few values will be a little higher or lower than that range.

Handling extreme outliers

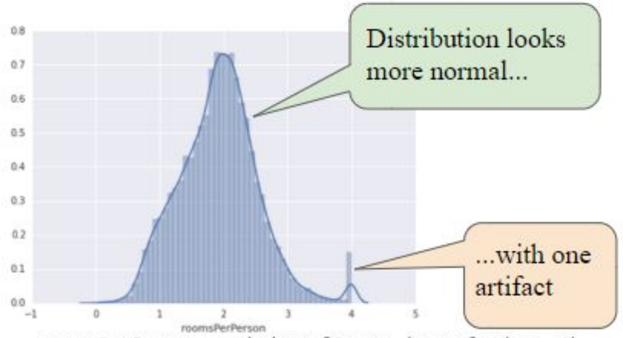


```
roomsPerPerson = totalRooms / population
```



```
roomsPerPerson = log((totalRooms / population) + 1)
```

Handling extreme outliers



```
roomsPerPerson = min(totalRooms / population, 4)
```

Logarithmic scaling still leaves a tail. Clipping the feature value at 4.0 doesn't mean that we ignore all values greater than 4.0. Rather, it means that all values that were greater than 4.0 now become 4.0.

This explains the funny hill at 4.0. Despite that hill, the scaled feature set is now more useful than the original data.

Scrubbing

Until now, we've assumed that all the data used for training and testing was trustworthy. In real-life, many examples in data sets are unreliable due to one or more of the following:

- Omitted values. For instance, a person forgot to enter a value for a house's age.
- Duplicate examples. For example, a server mistakenly uploaded the same logs twice.
- Bad labels. For instance, a person mislabeled a picture of an oak tree as a maple.
- Bad feature values. For example, someone typed in an extra digit, or a thermometer was left out in the sun.
-

Scrubbing

Once detected, you typically "fix" bad examples by removing them from the data set. To detect omitted values or duplicated examples, you can write a simple program. Detecting bad feature values or labels can be far trickier.

In addition to detecting bad individual examples, you must also detect bad data in the aggregate. Histograms are a great mechanism for visualizing your data in the aggregate. In addition, getting statistics like the following can help:

- Maximum and minimum
- Mean and median
- Standard deviation

Know your data

Follow these rules:

Keep in mind what you think your data should look like.

Verify that the data meets these expectations (or that you can explain why it doesn't).

Double-check that the training data agrees with other sources (for example, dashboards).

Treat your data with all the care that you would treat any mission-critical code. Good ML relies on good data.

3.

Feature Crosses

Kinds of feature crosses

We can create many different kinds of feature crosses. For example:

- [A X B]: a feature cross formed by multiplying the values of two features.
- [A x B x C x D x E]: a feature cross formed by multiplying the values of five features.
- [A x A]: a feature cross formed by squaring a single feature.

Thanks to stochastic gradient descent, linear models can be trained efficiently. Consequently, supplementing scaled linear models with feature crosses has traditionally been an efficient way to train on massive-scale data sets.

Kinds of feature crosses

A one-hot encoding of each generates vectors with binary features that can be interpreted as country=USA, country=France or language=English, language=Spanish. Then, if you do a feature cross of these one-hot encodings, you get binary features that can be interpreted as logical conjunctions, such as:

country:usa AND language:Spanish

As another example, suppose you bin latitude and longitude, producing separate one-hot five-element feature vectors. For instance, a given latitude and longitude could be represented as follows:

binned_latitude = [0, 0, 0, 1, 0]

binned_longitude = [0, 1, 0, 0, 0]

Suppose you create a feature cross of these two feature vectors:

binned_latitude X binned_longitude

Linear learners scale well to massive data. Using feature crosses on massive data sets is one efficient strategy for learning highly complex models.

Neural networks provide another strategy.

Moving on to Keras