Data Preparation

INFO-4604, Applied Machine Learning

University of Colorado Boulder

October 18, 2018

Prof. Michael Paul

Data Preprocessing

Preprocessing refers to the step of of processing your raw data in a way that makes it suitable for use in a learning algorithm.

- (in contrast to “processing” which would refer to the process of feeding the data into the learning algorithms)

When we talk about training data (or test data), there’s an assumption that it’s been preprocessed.

Data Preprocessing

The main components of preprocessing are:

- Getting features out of raw (unprocessed) data
  - To be covered in its own lecture
- Setting the values of the features
  - Fixing incorrect or missing values
  - Converting categorical values to numerical
  - Standardizing/normalizing the values to a common range
- Selecting which instances to include

Feature Extraction

Feature extraction is the process of getting the values of features out of raw data.

Example: in HW2, the instances were tweets. The “raw data” for each tweet is just a string.

The features were words, with values indicating how many times a word was in the tweet.

`sklearn` converts the strings into feature vectors for you.

- This requires *tokenizing* the strings (getting words separated by white space)

Feature Extraction

Different types of data and different tasks will require different types of features and different methods for obtaining features.

- More on this next time – for now, understand that feature extraction is usually the first step.

Not all datasets require feature extraction.

If the data is already organized into columns, you will usually take those variables to be your features.

Feature Values

Usually, at least some work needs to be done to transform the values of your features.

- Fixing incorrect or missing values
- Converting categorical values to numerical
- Standardizing/normalizing the values to a common range

Missing Values

Example: Some patients might not have had their heart rate recorded during a visit

PatientID	BP(S)	BP(D)	Heart Rate	Temperature
1234	120	80	75	98.5
1234	125	82		98.7
1245	140	93	95	98.5
3046	112	74	80	98.6

Missing Values

It’s surprisingly hard to deal with missing values.

- You can’t just “leave it out” of the learning algorithm – the math expects each feature to have a value.
- You can’t just set it to 0 – this means it is known to be 0, which is different from being unknown (especially if numerical).

Missing Values

Example: Some patients might not have had their heart rate recorded during a visit

PatientID	BP(S)	BP(D)	Heart Rate	Temperature
1234	120	80	75	98.5
1234	125	82		98.7
1245	140	93	95	98.5
3046	112	74	80	98.6

If only a small number of instances have missing values, maybe just remove those instances.

(don’t have to deal with the problem)

Missing Values

Example: Some patients might not have had their heart rate recorded during a visit

PatientID	BP(S)	BP(D)	Heart Rate	Temperature
1234	120	80	75	98.5
1234	125	82		98.7
1245	140	93	95	98.5
3046	112	74	80	98.6

If a lot of values are missing for a feature, maybe remove that feature.

(don’t have to deal with the problem)

Missing Values

Example: Some patients might not have had their heart rate recorded during a visit

PatientID	BP(S)	BP(D)	Heart Rate	Temperature
1234	120	80	75	98.5
1234	125	82	76.58?	98.7
1245	140	93	95	98.5
3046	112	74	80	98.6

You might also **impute** the missing values.

- Then you can treat the instances/features normally, and hopefully it's close enough.

Missing Values

Simplest methods to imputing a missing value:

- Take the mean of the values (if numerical)
- Take the majority value (if categorical)

You can also be more intelligent about it, but it depends on the data/task

- In the example of patient records, if there are multiple records for a patient, you could take the average value for that specific patient instead of averaging from all patients.

Missing Values

Example: Some patients might not have had their heart rate recorded during a visit

PatientID	BP(S)	BP(D)	Heart Rate	Temperature
1234	120	80	75	98.5
1234	125	82	UNK	98.7
1245	140	93	95	98.5
3046	112	74	80	98.6

You can also have a special “unknown” value.

- The classifier will then learn what to do when a feature has an unknown value.

Incorrect Values

A feature may have an incorrect value for various reasons.

- Transcription error (especially if automated, e.g. OCR)
- Human error (e.g., accidentally overwriting a value)
- Output error (e.g., if your feature is generated by another script, which had an error)

Incorrect Values

Some errors are easy to spot!

PatientID	BP(S)	BP(D)	Heart Rate	Temperature
1234	120	80	75	98.5
1234	125	82	720	98.7
1245	140	93	95	98.5
3046	112	74	80	98.6

Good to check for values outside of an accepted range (e.g., physical limitations, constraints in a system)

Incorrect Values

There are many techniques for **outlier** detection

- Outliers = “extreme” values
- Outlier values may be errors (though not necessarily)

Commonly accepted definition of an outlier is a value more than 2 standard deviations above or below the mean.

Visualizing the distribution of values can help you visually identify outliers.

Incorrect Values

Some errors are impossible to spot!

PatientID	BP(S)	BP(D)	Heart Rate	Temperature
1234	120	80	75	98.5
1234	125	82	72	98.7
1245	140	93	95	98.5
3046	112	74	80	98.6

Maybe a nurse transcribed this heart rate incorrectly (it was actually 73)

- No way we could know this from the data alone
- But be aware that data can have mistakes

Incorrect Values

Once you identify an incorrect value, you can treat it the same as a missing value
(if it’s incorrect, then the value is unknown)

Options:

- Remove from dataset
- Impute the value
- Give it a special “unknown” value

Categorical Values

What to do when features aren’t numeric?

PatientID	Sex	BP(S)	BP(D)	Heart Rate	Temperature
1234	Female	120	80	75	98.5
1234	Female	125	82	72	98.7
1245	Male	140	93	95	98.5
3046	Male	112	74	80	98.6

Categorical Values

What to do when features aren’t numeric?

PatientID	Sex	BP(S)	BP(D)	Heart Rate	Temperature
1234	Female	120	80	75	98.5
1234	Female	125	82	72	98.7
1245	Male	140	93	95	98.5
3046	Male	112	74	80	98.6

$\mathbf{x}_1$ = <“Female”, 120.0, 80.0, 75.0, 98.5>

How would we plug $\mathbf{x}_1$ into $\mathbf{w}^T\mathbf{x}_1$?

Categorical Values

Feature values need to be numeric for most learning algorithms! (decision trees an exception)

Common approach: **one-hot** encoding

Example: *Sex*

- Replace *Sex* with two variables:
    - *SexIsMale*, *SexIsFemale*
- These two variables are binary-valued
    - 1 if true, 0 if not

Categorical Values

Original encoding:

PatientID	Sex	BP(S)	BP(D)	Heart Rate	Temperature
1234	Female	120	80	75	98.5
1234	Female	125	82	72	98.7
1245	Male	140	93	95	98.5
3046	Male	112	74	80	98.6

One-hot encoding:

PatientID	M	F	BP(S)	BP(D)	Heart Rate	Temperature
1234	0	1	120	80	75	98.5
1234	0	1	125	82	72	98.7
1245	1	0	140	93	95	98.5
3046	1	0	112	74	80	98.6

Categorical Values

If you have *ordinal* values (e.g, small, medium, large), you might encode them with one feature that has increasing numerical values (e.g., 1, 2, 3)

- See book more for examples

Also note: you might have values consisting of numbers, but should still be treated as categorical instead of numerical values

- (e.g., zip code 80309)

Normalization

Normalization (or **standardization**) is the process of adjusting values so that the values of different features are on a common scale.

- Often these terms are used interchangeably
- The book distinguishes them as:
  - Normalization puts values in the range of [0,1]
    - When working with probabilities, “normalization” refers to converting values to probabilities.
  - Standardization converts values to their *standard score*

Normalization

Min-max normalization adjusts values as:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

where X_{min} is the smallest value of the feature, and X_{max} is the largest.

This converts all values to the range [0, 1], where the smallest value will be 0 and largest value will be 1.

Can also map to the range [a, b] using:

$$X' = a + \frac{(X - X_{min})(b - a)}{X_{max} - X_{min}}$$

Normalization

Standard score (or **z-score**) normalization adjusts values as:

$$X' = \frac{X - \mu}{\sigma}$$

where μ is the mean value of that feature, and σ is the standard dev.

Negative z-scores are values that are below the mean, while positive z-scores are above the mean, and the mean has a z-score of 0.

A z-score of 1 or -1 is one standard deviation above or below the mean.

Removing Instances

In some cases, you may want to remove certain instances from your dataset.

Primary reasons:

- de-duplication
- low-quality instances

Removing Instances

You may want to remove duplicates (identical or very similar instances) from your training data.

- Your classifier might be biased toward learning patterns that are more common in the duplicated instances.
- Not a necessary step, but sometimes useful.

Removing Instances

You may want to remove instances that you suspect are incorrectly labeled.

Common reasons:

- Annotated by someone you determined to be a poor annotator
- Instances you suspect are not meaningful (e.g., one-word reviews in a dataset of product reviews)

Conclusion

Most machine learning algorithms assume the input **x** is correct and complete

- So make sure that assumption is true by cleaning your data

“Garbage in, garbage out”