

Missing features

What if our data is not perfect ?

What do we do about examples with missing features

- training examples
- test examples

We have thus far been treating this as an annoyance

- problem is important
- far from simple

The simplest thing to do would be to drop the example

- can't do it with a test example
- reduces amount of data in training

We could alternatively, drop the feature entirely

- the features missing among several examples may be disjoint

Either way, losing training data is not desirable, particularly for small datasets

- We will motivate the problem and illustrate the issues
- We will examine naive solutions
 - almost always bad !
- We will show an interesting solution using Random Forests
- Preview of clustering methods

The term **imputation** refers to creating a substitute for the missing value of a feature in one example.

To frame our discussion

Let

- Let f denote the index of a feature
- $\mathbf{x}^{(m')}$ be an example (either training or test) with missing feature $\mathbf{x}_f^{(m')}$
- $\hat{\mathbf{x}}^{(m')}$ be the imputed value for $\mathbf{x}^{(m')}$

As usual let \mathbf{X}, \mathbf{y} be our labeled training examples.

$$\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) | 1 \leq i \leq m\}$$

Naive methods for imputation

Magic numbers

Let's start with a truly awful method: set $\hat{\mathbf{x}}_f^{(m')}$ to a "magic number"

- 0
- -999

Why is the magic number awful ?

Consider a training set representing the population of NYC, with features Weight, Height, Age, etc.

Suppose Weight, which is at index f of example vector $\mathbf{x}^{(m')}$ is missing.

Setting $\hat{\mathbf{x}}_f^{(m')} = 0$ is awful because the imputed value is not likely

- $p(\mathbf{x}_f^{(m')} = 0) \approx 0$

Mean, median, percentile

How about something more likely, like the mean or median ?

Better

- $p(\mathbf{x}_f^{(m')} = \bar{\mathbf{x}}_f) > 0$

Still not perfect.

- What if $p(\mathbf{x}_f)$ were a bi-modal distribution
 - lots of examples with extreme values, few in the middle

So mean and median are better than magic numbers in many situations but not all.

Even worse:

Suppose example $\mathbf{x}^{(m')}$ is an infant: is $\bar{\mathbf{x}}_f$ still reasonable?

- $p(\mathbf{x}_f^{(m')} = \bar{\mathbf{x}}_f | \mathbf{x}_{\text{Age}}^{(m')} < 1) \approx 0$

So the mean, median etc.

- provides a reasonable imputation in a univariate sense
- provides a less reasonable imputation in a multivariate sense
 - conditional on other features like Age

Imputation depends on how the imputed value is used

Less obvious is that $\hat{\mathbf{x}}_f^{(m')}$ will be used for training some model.

So perhaps we should consider how the model that we are going to fit uses $\hat{\mathbf{x}}_f^{(m')}$.

To illustrate, suppose the model uses the dot product to measure similarity among examples

- a variant of KNN

Knowing this, we can show that different choices of $\hat{\mathbf{x}}_f^{(m')}$ influence the similarity metric.

In [6]:

```
df  
  
print("A vs B")  
compare_subs(A,B)  
  
print("A vs C")  
compare_subs(A,C)
```

Out[6]:

	a	d	e	b	c	f
A	4.0	5.0	1.0	NaN	NaN	NaN
B	5.0	NaN	NaN	5.0	4.0	NaN
C	NaN	2.0	4.0	NaN	NaN	5.0

A vs B

Substitute 0: similarity= 0.38
Substitute w/feature mean: similarity= 0.94
Center, then Substitute 0: similarity= 0.09

A vs C

Substitute 0: similarity= 0.32
Substitute w/feature mean: similarity= 0.87
Center, then Substitute 0: similarity= -0.56

- A versus B
 - missing values are paired against relatively high values
 - Substituting 0 (a low value) reduces similarity
 - Substituting mean (a relatively high value) increase similarity
 - A is a tougher rater: $A.mean() < B.mean()$
 - in the end, A and B had only a *single* true point of comparison ("a")
 - you made up the similarity
- A versus C:
 - NO feature "a" in common !
 - But at least A and B are closer than A and C for some substitutions

Cosine similarity

- is a *scale dependent* measure
 - so centering, scaling matter
 - Analogy
 - difference between Covariance and Correlation
 - Correlation is Covariance of normalized (scaled) variables

The choice of $\hat{\mathbf{x}}_f^{(m')} = 0$ is **not** neutral if the data is not centered

- e.g., if 0 is the minimum of \mathbf{x}_f , rather than the average

Predictive methods for imputation

Hopefully the preceding examples illustrated some issues in imputation.

Can we do better ?

Let $\mathbf{x}_{\bar{f}}$ denote the vector of features *excluding* the one at index f .

We can frame the imputation problem as finding

$$p(\mathbf{x}_f^{(m')} | \mathbf{x}_{\bar{f}}^{(m')})$$

That is: find likely values for the missing feature, *given* values for the non-missing features.

How do we do this ?

Machine Learning to the rescue !

- fit a model on the subset of training examples
 - that *have* feature f (used as target)
- use the model to predict $\hat{\mathbf{x}}_f^{(m')}$

Simple predictive imputation

Some ideas

- Naive Bayes
 - Assumption of distribution of features can compensate for missing features
- Regression
 - $\mathbf{x}_f = \Theta^T \mathbf{x}_{\bar{f}}$
 - feature f as a function of the other features
 - $\mathbf{x}_f = \Theta^T \mathbf{z}$
 - \mathbf{z} may be features, not present in \mathbf{x} , that are believed to be correlated with \mathbf{x}_f

Proximity based imputation

A proximity based method

- creates a proximity (opposite of distance) measure $prox(\mathbf{x}^{(m')}, \mathbf{x}^{(i)})$ between $\mathbf{x}^{(m')}$ and training example $\mathbf{x}^{(i)}$
- $\hat{\mathbf{x}}_f^{(m')}$ is the proximity weighted average of the values of the feature in the training set

$$\hat{\mathbf{x}}_f^{(m')} = \sum_{i=1, i \neq m'}^m prox(\mathbf{x}^{(m')}, \mathbf{x}^{(i)}) \mathbf{x}_f^{(i)}$$

That is

- the missing value should be similar to the feature value in training examples "similar" to $\mathbf{x}^{(m')}$.

The definition of proximity (similarity) will vary.

Note

For categorical $\mathbf{x}_f^{(m')}$ use the most frequent non-missing value

- where the frequency is weighted by proximities.

The method works for multiple missing features but we illustrate with a single one for simplicity.

Limitations

For the imputation of

$$p(\mathbf{x}_f^{(m')} | \mathbf{x}_{\bar{f}}^{(m')})$$

we are implicitly assuming that if feature vectors $\mathbf{x}^{(i)}$, $\mathbf{x}^{(i')}$ are "similar" then so are their targets

$$\mathbf{y}^{(i)} \approx \mathbf{y}^{(i')}$$

With that limitation in mind there are related methods

- Clustering
 - find groups of examples with common features
 - K-means
 - PCA, Recommender systems
 - Unsupervised Machine Learning: Preview of coming lecture !

Random Forest proximity method for missing data

There is an interesting method for using a Random Forest to impute missing data.

It is interesting because proximity is determined by both the features *and* the target so we are modeling

$$p(\mathbf{x}_f | \mathbf{y}, \mathbf{x}_{f^-})$$

That is, it fits a model of \mathbf{x}_f given the features other than f **and** the target.

A test example with missing features doesn't have a target; we will see how this method adapts.

Missing feature in Training example

We will use a Random Forest to define a proximity measure.

[Missing Value Imputation using Random Forest](https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm#missing1)

[.https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm#missing1](https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm#missing1)

- Initialization
 - Set $\hat{\mathbf{x}}_f^{(m')}$ to a "reasonable" guess
 - Continuous: mean, median
 - Categorical: most frequent
 - Create the initial Random Forest $F_{(0)}$

- Iteration i :
 - Define the proximity to example $\mathbf{x}^{(i)}$
 - $prox(\mathbf{x}^{(m')}, \mathbf{x}^{(i)}) = \#$ of trees in $F_{(i-1)}$ with $\mathbf{x}^{(m')}$,
 $\mathbf{x}^{(i)}$ in same leaf
 - Update imputed value $\hat{\mathbf{x}}_f^{(m')}$

$$\hat{\mathbf{x}}_f^{(m')} = \sum_{i=1}^m prox(\mathbf{x}^{(m')}, \mathbf{x}^{(i)}) \mathbf{x}_f^{(i)}$$
 - Create next Random Forest $F_{(i)}$

Iterate until convergence.

The authors suggest 4-6 iterations suffice.

Missing feature in Test example

Method similar to that for a missing feature in a Training example, once we deal with a crucial difference

- there is **no label** for the test example (that's what we're trying to predict)

Suppose the classification target $\mathbf{y} \in C$ (i.e., the possible labels)

$$C = \{c_k | 1 \leq k \leq |C|\}$$

- For each $c \in C$:
 - $\hat{\mathbf{x}}_{f,c}^{(m')}$ is the imputed value obtained from the above by assuming $\mathbf{y}^{(m')} = c$
 - that is, run the missing feature for training algorithm assuming label of $\mathbf{x}^{(m')}$ is c

We now have one imputed value $\hat{\mathbf{x}}_{f,c}^{(m')}$ and final Random Forest per class c .

Which one do we choose ?

Observe that the c^{th} Random Forest *should* predict class c given input $\mathbf{x}^{(m')}$ since we set $\mathbf{y}^{(m')} = c$

So we choose the forest and imputed value $\hat{\mathbf{x}}_{f,c}^{(m')}$

- from the class c in which $\mathbf{x}^{(m')}$ is most often classified as being in class c .

Now casting

The field of economic forecasting encounters a problem similar to missing data

- many economic indices are combinations of sub-indices
 - sub-indices published at different frequencies
 - sub-indices published on different days

The "total" index can't be computed until **all** sub-indices have been released.

So with respect to an "early" publication date, some sub-index data is missing.

Now-casting (a play on "forecasting") uses techniques to make early predictions of sub-index values

In some cases they use features \mathbf{z} believed to be correlated with actual features \mathbf{x} :

- derive higher frequency values for low frequency data (annual GDP, monthly Manufacturing)
 - National Manufacturing employment may be highly correlated to Employment in a few states
 - state-level employment may be published
 - at higher frequency
 - earlier date
 - Many of these low frequency features are *composites* of other features
 - some elements of the composite are released before others
 - may predict the whole
 - Now-casting site (<https://www.now-casting.com/home>).

Missing data imputation in sklearn

- SimpleImputer, IterativeImputer (<https://scikit-learn.org/stable/modules/impute.html>)
 - IterativeImputer with different regression estimators (https://scikit-learn.org/stable/auto_examples/impute/plot_iterative_imputer_variants_compariso)
-

In [7]: `print("Done")`

Done