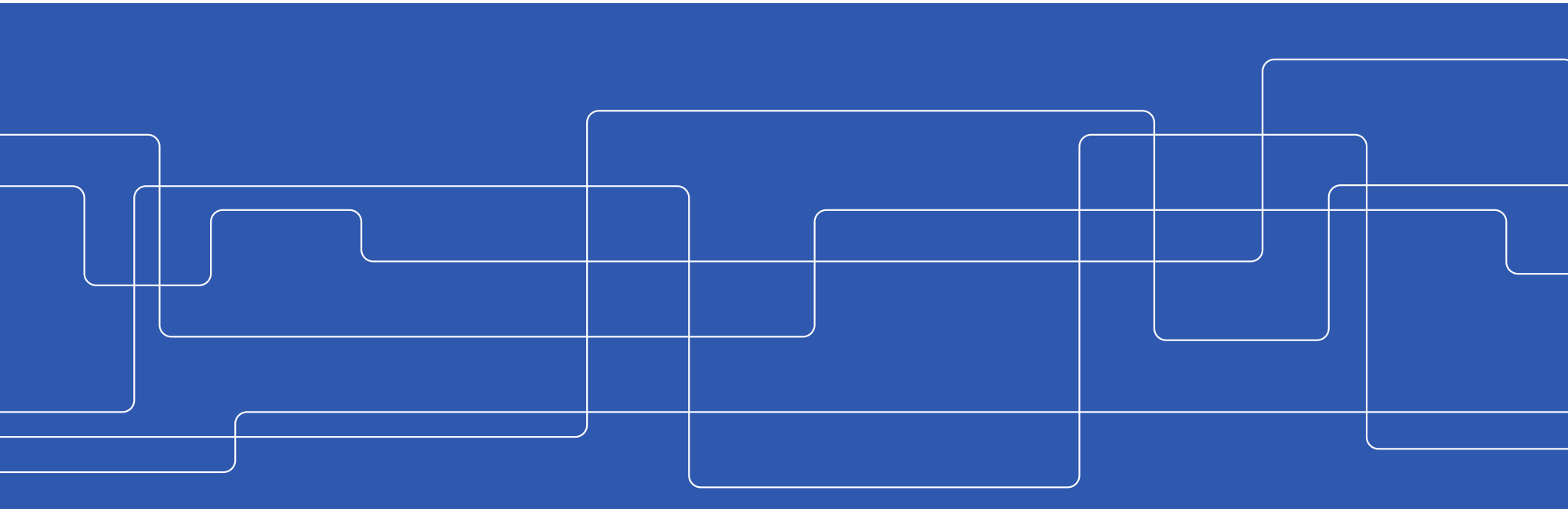
ID2214/FID3214 2020

Programming for Data Science – Data Preparation

Henrik Boström

Prof. of Computer Science - Data Science Systems
Division of Software and Computer Systems
Department of Computer Science
School of Electrical Engineering and Computer Science
KTH Royal Institute of Technology
bostromh@kth.se



Outline

Data preparation

Handling missing values

Encoding features

Dimensionality reduction

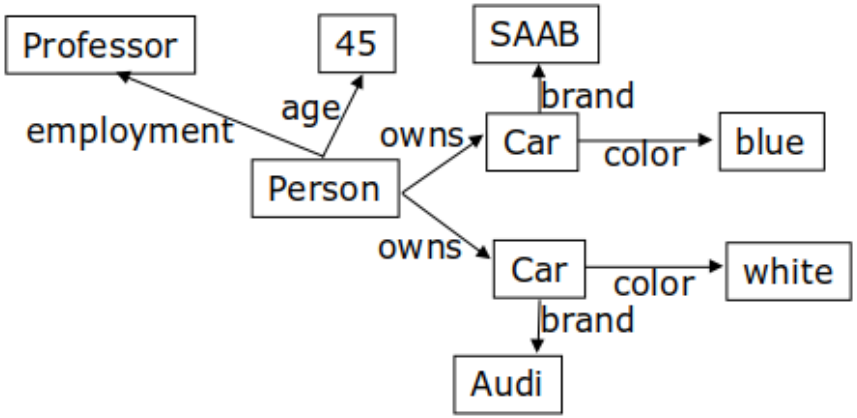
Example examination questions

Data preparation

The way in which data has to be prepared depends on the goal of the analysis and the algorithms that will be employed. Such requirements may include:

- the instances need to be represented by fixed-length feature vectors
- for predictive modeling, labels have to be assigned to instances, and information from test instances should not affect choice of data preparation and learning algorithms
- there can be no missing, numerical or categorical values
- numerical features have to be normalized
- the *curse of dimensionality* has to be remedied by limiting the number of features

Representation



Empl.	Age	Brand1	Color1	Brand2	Color2	Brand3	Color3	...
Prof.	45	SAAB	blue	Audi	white	-	-	...

Empl.	Age	No. SAABs	No. Audis	No. Volvos	...	No. blues	No. whites	No. greens	...
Prof.	45	1	1	0	...	1	1	0	...

Handling missing values

- Some techniques for analyzing data can deal directly with missing data; no need for special handling
- Other techniques require missing values to be handled, by
  - removing them, i.e., removing rows and/or columns, or
  - replacing (*imputing*) them
- How to impute missing values is a research area of its own; with techniques ranging from replacing missing values with the mean or mode to more advanced methods relying on using values from nearest neighbors

Handling missing values in DataFrames

- Dropping rows or columns with missing values

```
df = pd.DataFrame({"id":[np.nan,2,3,4,5],
                   "grade":[np.nan,"b",np.nan,"c",np.nan],
                   "award":[np.nan, "gold", "silver",
                   "bronze", np.nan]})
df.dropna(how="any")          # axis="index" is default
                              # alt. use axis="columns"
    id grade   award
1  2.0     b    gold
3  4.0     c  bronze
df.dropna(how="all",subset=["grade","award"])
    id grade   award
1  2.0     b    gold
2  3.0   NaN  silver
3  4.0     c  bronze
```

Handling missing values in DataFrames (cont.)

- Imputing missing values

```
df = pd.DataFrame({"id":[np.nan,2,3,4,5],
                   "grade":[np.nan,"b",np.nan,"c",np.nan],
                   "award":[np.nan, "gold", "silver",
                   "bronze", np.nan]})
values = {"grade": "e", "award": "iron"}
df.fillna(value=values)
```

```
    id grade   award
0  NaN     e    iron
1  2.0     b    gold
2  3.0     e  silver
3  4.0     c  bronze
4  5.0     e    iron
```

```
df["id"].fillna(df["id"].mean(),inplace=True)
df["award"].fillna(df["award"].mode()[0],inplace=True)
```

Encoding features: from numerical to categorical (*Discretization*)

- The standard procedure to turn numerical (continuous) feature values into categorical is through *binning*, i.e., ranges of values define categories.
- Binning is often by either *equal width*, i.e., each range is the same size, or by *equal size*, i.e., the same number of observed values fall into each range
- Binning can of course be done using user-specified bins (of different width and/or sizes) and may involve multiple features (*grid-based binning*).

Binning in DataFrames

- Equal-width binning (using cut)

```
df = pd.DataFrame({"values": np.random.rand(100)})
res, bins = pd.cut(df["values"],10,retbins=True)
bins
array([0.02501036, 0.12329601, 0.22060853, 0.31792105,
       0.41523357, 0.5125461 , 0.60985862, 0.70717114,
       0.80448366, 0.90179619, 0.99910871])
res
0      (0.61, 0.707]
1     (0.415, 0.513]
2     (0.221, 0.318]
3     (0.123, 0.221]
...
new_res = pd.cut(df2["values"],bins)
```

Binning in DataFrames (cont.)

- Equal-sized binning (using qcut)

```
df = pd.DataFrame({"values": np.random.randn(100)})
res, bins = pd.qcut(df["values"],10,retbins=True,
                    labels=list("abcdefghij"))
bins
array([-2.52042001, -1.28725864, -0.85326262, -0.56196045,
       -0.37714909, -0.09507675,  0.31009968,  0.60596668,
        0.81044397,  1.29253893, 2.45534755])
res
0     j
1     b
2     b
3     j
```

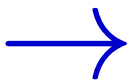
Encoding features: from categorical to numerical

- It is typically not a good idea to turn a (multi-valued) categorical feature into a single numerical feature (unless the values have an ordering)
- A common approach is to employ *one-hot encoding*, i.e., a new (binary) feature is created for each possible categorical value, and the new feature values for an instance (row) are all assigned zero except for the feature corresponding to the categorical value appearing in the original row (which is assigned one).
- Note that one-hot encoding may lead to *feature explosion*, i.e., the number of features growing beyond control. Grouping of values or other *dimensionality reduction* techniques may be required.

Encoding features: from categorical to numerical (cont.)

- One-hot encoding (example)

	value
0	a
1	b
2	c
3	a
4	b
5	c



	value-a	value-b	value-c
0	1	0	0
1	0	1	0
2	0	0	1
3	1	0	0
4	0	1	0
5	0	0	1

Encoding features: normalization

- min-max normalization; $x_i' = \frac{x_i - x_{min}}{x_{max} - x_{min}}$

```
df = pd.DataFrame({"values": np.random.randn(100)})
min = df["values"].min()
max = df["values"].max()
df["values"] = [(x-min)/(max-min) for x in df["values"]]
```

- z-normalization; $x_i' = \frac{x_i - \bar{x}}{s}$

```
df = pd.DataFrame({"values": np.random.randn(100)})
mean = df["values"].mean()
std = df["values"].std()
df["values"] = df["values"].apply(lambda x: (x-mean)/std)
```

Encoding features: dimensionality reduction

- Feature selection
  - Filtering approaches, i.e., ranking the input features based on their correlation with the output (target) feature, using e.g., information gain, χ^2
  - Wrapper approaches, i.e., iteratively choosing features based on their effect/presence in models generated by a specific learning algorithm
- Principal Component Analysis (PCA), i.e., projecting multiple numerical features into new features by a linear combination, ordered by the amount of (remaining) variability they can account for, from which the (k) highest ranked are chosen; computationally costly ($O(p^2n + n^3)$)
- Random Projection (RP), i.e., projecting multiple numerical features into (k) new features using a (sparse) random matrix; computationally not so costly ($O(pnk)$)

Feature selection by filtering

- Selection of top-ranked categorical features

```
for col in df.columns:
        df[col] = df[col].astype("category")
res = [(col,[g.groupby("class").size().values
             for (n,g) in df.groupby(col)])
             for col in df.columns.drop("class")]
scores = [(col,score(r)) for (col,r) in res]
sorted_scores = sorted(scores,key=lambda tup: tup[1],
                       reverse=True)
filtered = [col for (col,score) in sorted_scores[:2]]
new_df = df.loc[:,filtered]
```

Summary

- We have covered various approaches to handling missing values and encoding features, i.e., changing the type and number of features
- A proper choice of feature encoding technique(s) may not only lead to that requirements of specific learning algorithms are met, but also to improved efficiency and/or effectiveness.

Example examination questions

1. Will mean-value imputation have the same effect, if performed before normalization, on the distribution of the normalized values for those values that were originally not missing, for min-max normalization and z-normalization? Explain your reasoning.

2. Will the use of min-max normalization prior to the use of equal-sized binning have any effect on model building compared to just using equal-sized binning (without normalization)? Explain your reasoning.