



COMPSCI 389

Introduction to Machine Learning

Days: Tu/Th. **Time:** 2:30 – 3:45 **Building:** Morrill 2 **Room:** 222

Topic 8.1: Data Processing

Prof. Philip S. Thomas (pthomas@cs.umass.edu)

Data Processing

- Data collection in the real world can be challenging
- Sometimes values are logged incorrectly
 - This can be hard to catch
 - A month into a project I worked on in industry, we found a bug in the data collection code that entirely corrupted the data we had been working with (and struggling with).
- Sometimes values are not logged or cannot be collected
 - This results in **missing data**
- Sometimes values arrive in forms that are harder to deal with (e.g., text), and should be converted into values that are easier to work with (e.g., integers).
- Sometimes values are poorly scaled

Missing Data

- **Question:** What can we do if some values are missing in the data set?
 - **Example:** Some students are missing exam scores.
- **Answer 1:** Remove rows with missing values.
 - This can add bias when there is a correlation between *when* points are missing and other features/labels.
 - This can be effective when only a few rows are missing values.
- **Answer 2:** Use **imputation techniques**.
 - Replace missing values with the mean or median feature value.
 - Replace missing values with the feature values from the nearest neighbor (or k nearest neighbors).
 - Use more sophisticated techniques to estimate the missing values.

Data Balancing

- Consider predicting whether a rock is a meteorite.
- Gather data by collecting 1 million rocks, and labeling as meteorite (1) or not a meteorite (0).
- Almost all will not be meteorites!
- A classifier that predicts 0 will perform nearly optimally.
- **Idea:** “Oversample” points from the minority class, simulating having more points of that type.
 - **Method:** Duplicate rows from the minority class (meteorite) until the two classes (meteorite / not meteorite) have an equal numbers of samples.

Data Format

- Categorical values are often easier to work with as discrete numerical values.
 - Categorical values can easily be replaced with integers.
- This can cause problems with nominal features
 - **Major:** “computer science” \rightarrow 0, “philosophy” \rightarrow 1, “physics” \rightarrow 2, “sociology” \rightarrow 3, etc.
 - Let this be the j^{th} feature.
 - A linear parametric model could place a weight w_j on this feature.
 - This suggests that there is meaning to the numbers assigned to categories, since the integer values are scaled by the weight.

One Hot Encoding

- **One hot encoding** is a common strategy to avoid assigning meaning to the encoding of categorical features.
- If the feature has m possible values, it is converted into m features.
 - One column is converted into m columns.
- The value of the j^{th} new feature is 1 if the original feature took its j^{th} value, and 0 otherwise.
- Example: Original feature: “red”, “green”, “blue”
 - Three new features, “is red”, “is green”, and “is blue”
 - If “red”, the three new features have values [1, 0, 0]
 - If “green”, the three new features have values [0, 1, 0]
 - If “blue”, the three new features have values [0, 0, 1]

One Hot Encoding (Python/Pandas)

- `get_dummies(DataFrame, columns)`
 - `DataFrame`: The DataFrame with one or more categorical columns that you want to one hot encode.
 - `Columns`: The columns in the data frame that you would like to one hot encode.
 - `Return value`: A new data frame with one hot encodings.

- **Example:**

```
import pandas as pd
one_hot_encoded_df = pd.get_dummies(df, columns=['major'])
```

- **Note:** `get_dummies` returns columns with “True” and “False” rather than 1 and 0. You can obtain the numerical values with the argument `dtype=float`.

Feature Scaling

- When features have very different scales, it can cause problems for some ML algorithms.
 - **Question:** Consider a data set with income (range 0 to 1 million) and age (range 0 to 100). If we use nearest neighbor algorithms with Euclidean distance, what will happen?
 - **Answer:** Points with (relatively) slightly different incomes will be viewed as far apart relative to points with different ages.
 - **Note:** This is not unique to nearest neighbors algorithms. *Most* ML algorithms can struggle when features have very different scales.
- When all features have a very large or small scale, it can change the necessary hyperparameters in unintuitive ways.
 - **Example:** The step size for running gradient descent to fit a linear parametric model, using the second-degree polynomial basis, to the GPA data set (see 8.0 Data Cleaning Intro.ipynb).

Feature Scaling

- **Idea:** Re-scale features.
- **Approach 1 (Min-Max Scaling):** Normalize to the range [0,1]
 - $x_{\text{normalized}} = (x_{\text{unnormalized}} - \text{min}) / (\text{max} - \text{min})$
 - Scikit-learn includes “Scalers” that perform common feature rescaling.
 - The `fit_transform` function “fits” the scaler to the data (e.g., calculating min and max values of features) and then “transforms” the data (applies the specified rescaling).

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df_scaled = scaler.fit_transform(df)
```

Feature Scaling

- **Idea:** Re-scale features.
- **Approach 2 (Standardization):**
 - Centers the feature (so the average is zero)
 - Rescales the feature so that the standard deviation is 1
 - $x_{\text{normalized}} = (x_{\text{unnormalized}} - \text{mean}) / (\text{standard deviation})$

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)
```

Feature Scaling

- There are *many* other strategies, but min-max scaling and standardization are the most common.
- Other examples:
 - Robust scaling (like standardization, but more robust to outliers)
 - `RobustScaler()`
 - Normalization (scales individual rows to have unit length)
 - `Normalizer()`
 - Max Abs Scaling (divides by `max(abs(x))`)
 - `MaxAbsScaler()`

Basis Functions

- **Note:** scikit-learn provides functions for applying the polynomial basis!

```
from sklearn.preprocessing import PolynomialFeatures

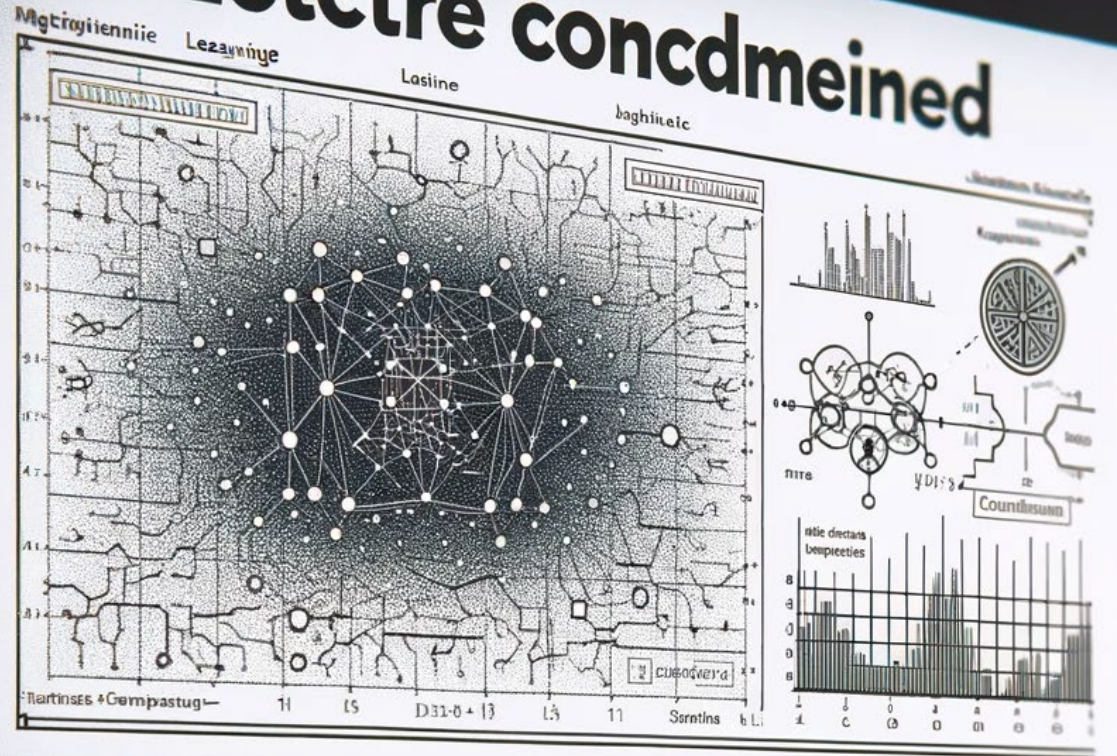
# Expand features into polynomial basis
poly = PolynomialFeatures(degree=polynomial_degree)
X_poly = poly.fit_transform(X_scaled)
```

See `8.2 Data Processing Example.ipynb`

- This performs gradient descent on the sample mean squared error, fitting a linear parametric model with the second-degree polynomial basis to the GPA data.
- By including feature scaling (standardization), it is now much easier to tune the step size!

End

Letctre concdmeined



Dggnnbnic



Mbcine Learning

Thank you.

