Tufts

Class #07:
Feature Engineering

Machine Learning (COMP 135)

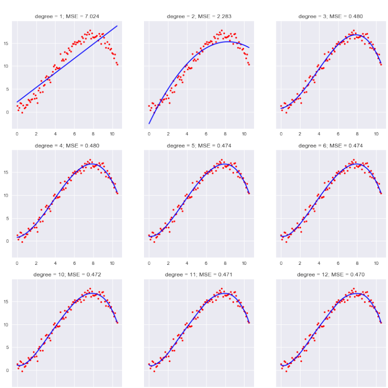
1

Feature Engineering

- As we saw with polynomial regression, we often want to ***transform*** our data in order to get better results from a machine learning algorithm
- We often get better results by:
  1. Changing how features are represented.
  2. Adding new features.
  3. Deleting/ignoring some features.

2

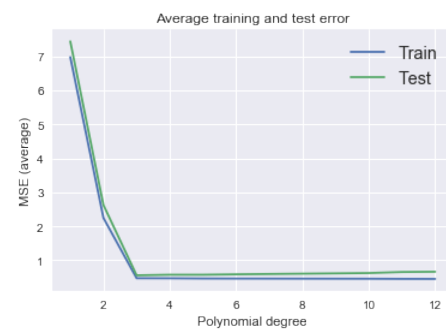
Example: Higher-Order Polynomial Features



- Transforming data by mapping to higher-degree polynomials, and then fitting a linear regression, can reduce error
  - Gains are most significant at first, and then error starts to level off

3

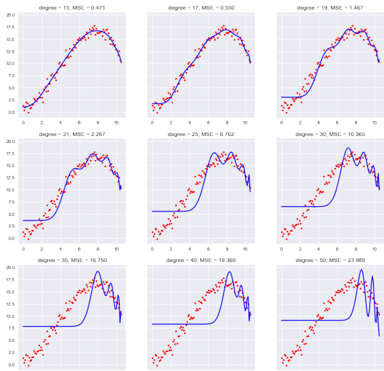
The Cost of Feature Transformation



- Not every transformation is as useful as others
- The polynomial degrees above 3 from previous slide also start to show some evidence of over-fitting, as revealed by cross-validation

4

The Cost of Feature Transformation



- Not every transformation is useful—at very high polynomials, some of the mathematics of the linear regression libraries in `sklearn` break down
  - Mathematically, we expect better and better fits
  - In practice, the method ceases working effectively, and models are generally useless

Issue: "Uneven" Features

- We apply some algorithm to data, fitting a linear (or other) function, for regression or classification
- ***Problem***: data-features have radically different ranges
  - E.g., data-set of health & demographic data
  - *age* in [0, 95], *height* in [0,7], *weight* in [0,300], *co-morbidities* in [0,2], *income* in [10K, 250K]
  - Algorithm needs to make coefficient on *income* very small, while others need to be made relatively large
- This can lead to very poor performance, and even failure of model to converge

Feature Rescaling

- ***Input***: numeric features, each with a minimum/maximum
  - E.g., A in [0, 1], B in [-5, 5], C in [-3333, -2222]
- ***Output***: transformed feature vector, all on same scale
- Each feature f now has values in range [0, 1]

$$\phi(x_n)_f = \frac{x_{nf} - \min_f}{\max_f - \min_f}$$

  - $\min_f$ = minimum value for f in training data
  - $\max_f$ = maximum value for f in training data

Feature Standardization

- ***Input***: numeric features, each on any scale
- ***Output***: transformed feature vector, all evenly distributed
- Each feature f now has zero mean and unit variance

$$\phi(x_n)_f = \frac{x_{nf} - \mu_f}{\sigma_f}$$

  - μ_f = empirical mean for f in training data
  - σ_f = empirical standard deviation for f in training data
- Features treated as if normally distributed
  - If original data is (approximately) normal, typical range is [-3, 3]
  - Also known as a z-score transform

Issue: Too Many Features

- We apply some algorithm to data, fitting a linear (or other) function, for regression or classification
- ***Problem***: many, many data-features
  - E.g., data-set of high-resolution (megapixel) images
  - Each pixel in an image has 3 numerical values corresponding to color channels
  - Algorithm needs to find coefficients for each such value
- Again, this can lead to very poor performance, and it may be impractical even to use all the features
  - We generally ***don't know*** which are truly important

One Solution: Best Subset Selection

- An algorithm for choosing a subset from F total features:
  1. Start with a ***null model*** M_0, predicting empirical mean
  2. For $k = 1, 2, \ldots, F$:
     - Fit all $\binom{F}{k}$ models, each using k features
     - Let M_k be the model from this selection with lowest error
  - Select a final, best-performing model from $M_0, \ldots, M_F$, using cross-validation, or some other technique

---

- Main issue: too many subsets
  - There are $O(2^F)$ such collections of features
  - For problems with large feature-sets, this proves infeasible

Another Approach: Forward Stepwise Selection

1. Start with a ***null model*** M_0, predicting empirical mean
2. For $i = 1, 2, \ldots, F$:
   - Build a model using only feature f_i
   - Store the best of these models as M_1
3. For each size $k = 2, \ldots, F$:
   - Add each possible feature not already included, to build (F – (k–1)) new models
   - Store the best of these models as M_k
4. Select a final, best-performing model from $M_0, \ldots, M_F$, using cross-validation, or some other technique

---

- When the number of features is much smaller than the overall data-set size, this is much more efficient
  - In general, for best results, we usually ***need*** this to be true anyhow…

Backwards Stepwise Selection

The basic forward process can also be run backwards:

1. Start with ***all*** features
2. Gradually test all models with one feature ***removed*** from each
3. Repeat to remove $2, 3, \ldots, F$ features, all the way down to single-feature models
4. Select the best-performing model seen overall

Best Subset vs. Stepwise Selection

#Variables	Best Subset	Forward Stepwise
1	`Age`	`Age`
2	`Age, Weight`	`Age, Weight`
3	`Age, Weight, Height`	`Age, Weight, Height`
4	`Age, Weight, Income, ZipCode`	`Age, Weight, Height, Income`

- Common cases exist where stepwise selection does not lead to the best possible feature-set
  - Since we add a feature at each step, but ***never remove*** any, stepwise process can get "locked in" on subsets of features
  - Best subset process considers ***all*** possible combinations, and so can add/remove features at any point