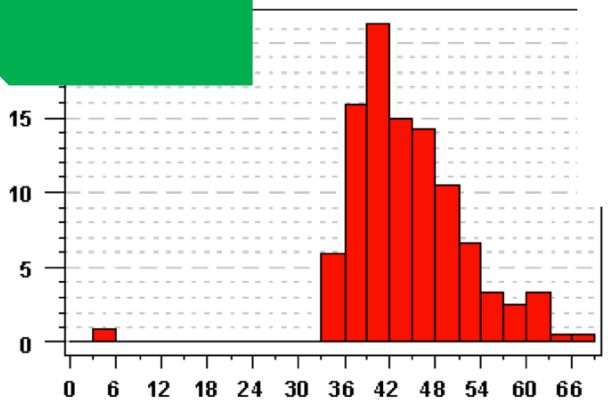
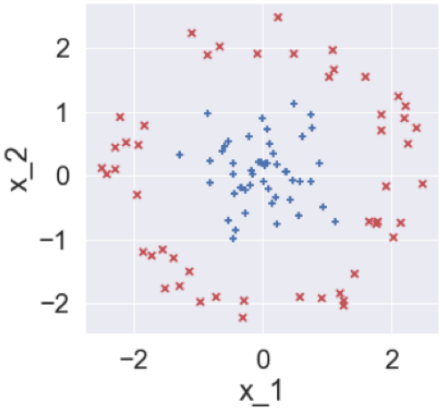
Tufts COMP 135: Introduction to Machine Learning

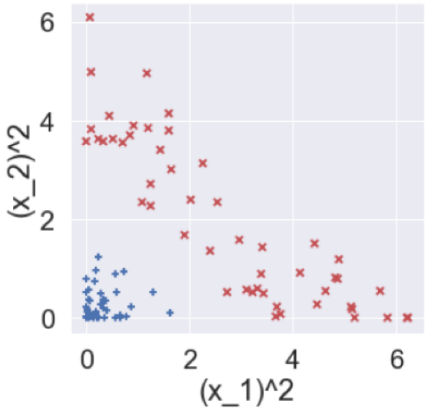
https://www.cs.tufts.edu/comp/135/2019s/

Feature Engineering

x **transform** $\phi(x)$







Many slides attributable to:
Erik Sudderth (UCI)
Finale Doshi-Velez (Harvard)
James, Witten, Hastie, Tibshirani (ISL/ESL books)

Prof. Mike Hughes

Logistics

- Project 1 is out! (due in two weeks)
  - Start early! Work required is about 2 HWs
- HW4 will be out next Wed
  - Due two weeks later (1 week after project)
  - More time to learn req'd material
- Class TOMORROW 3pm
  - Mon on Thurs at Tufts

Objectives Today: Feature Engineering

Concept Check-in

How should I preprocess my features?

How can I select a subset of important features?

What to do if features are missing?

Check-in Q1: logsumexp

```
def my_log_sum_exp(scores_K):
    return np.log(np.sum(np.exp(scores_K)))
```

```
my_log_sum_exp([0.0, 3.0, -1.0])
```

```
my_log_sum_exp([-100.0, -97.0, -101.0])
```

```
my_log_sum_exp([-1000.0, -997.0, -1001.0])
```

What scalar value should these calls produce?
What happens instead with a real computer? What is the fix?

logsumexp explained

$$\begin{aligned}
\text{logsumexp}([-100, -97, -101]) &= \log(e^{-100} + e^{-97} + e^{-101}) \\
&= \log(e^{-97}(e^{-3} + e^{0} + e^{-4})) \\
&= \log(e^{-97}) + \log\left(e^{-3} + e^{0} + e^{-4}\right) \\
&= -97 + \log\left(\underbrace{e^{-3} + e^{0} + e^{-4}}_{1 \leq \text{sum} \leq 3}\right)
\end{aligned}$$

Factor out the MAX of -97

Check-in Q2: Gradient steps

How can I diagnose step size choices?

What are three ways to improve step size selection?

Check-in Q2: Gradient steps

How can I diagnose step size choices?

Trace plots of loss, gradient norm, and parameters

Explore like “Goldilocks”, find one too small and one too big

What are three ways to improve step size selection?

Use decaying step size

Use line search to find step size that reduces loss

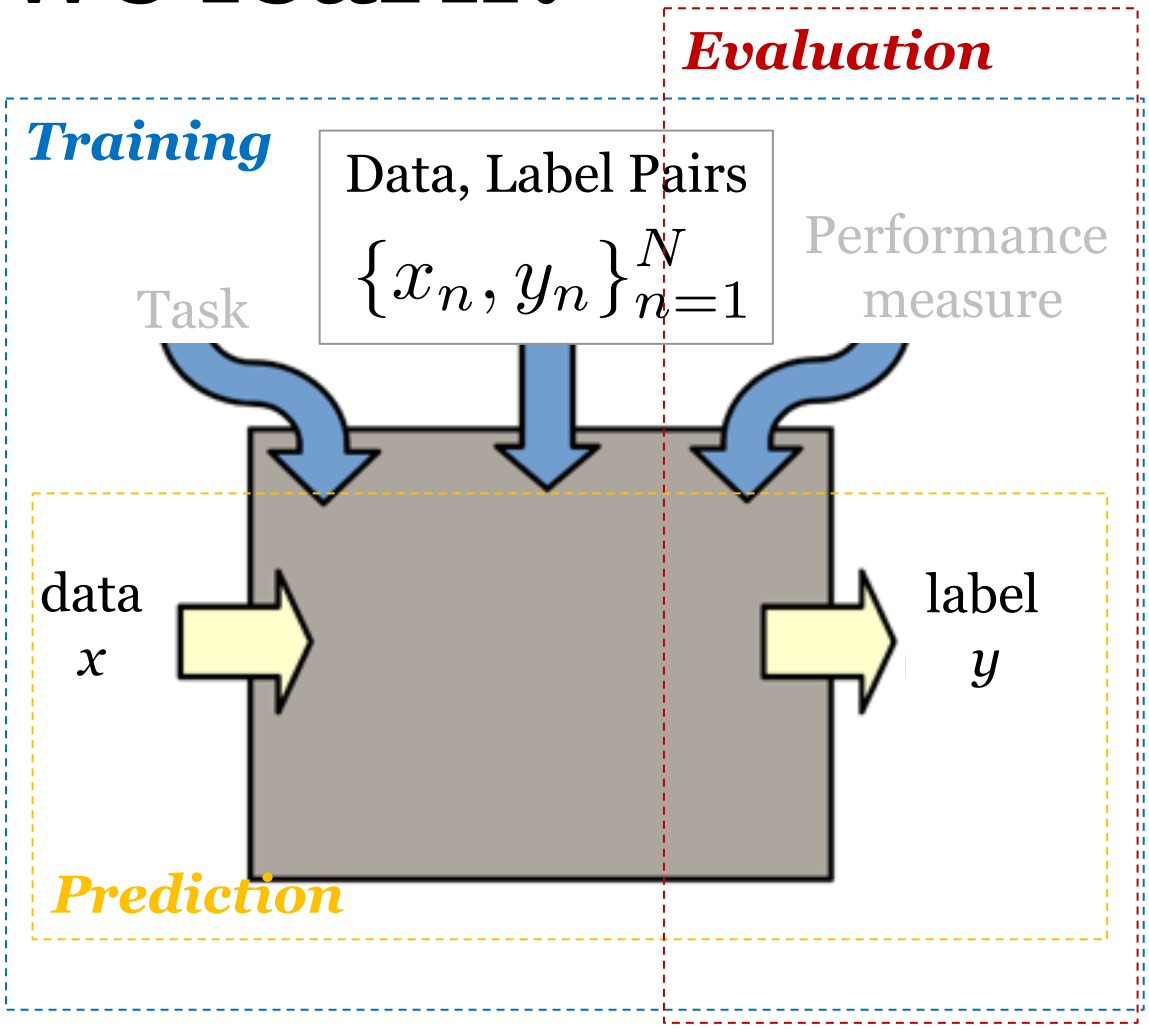
Use second order methods (Newton, LBFGS)

What will we learn?

Supervised Learning

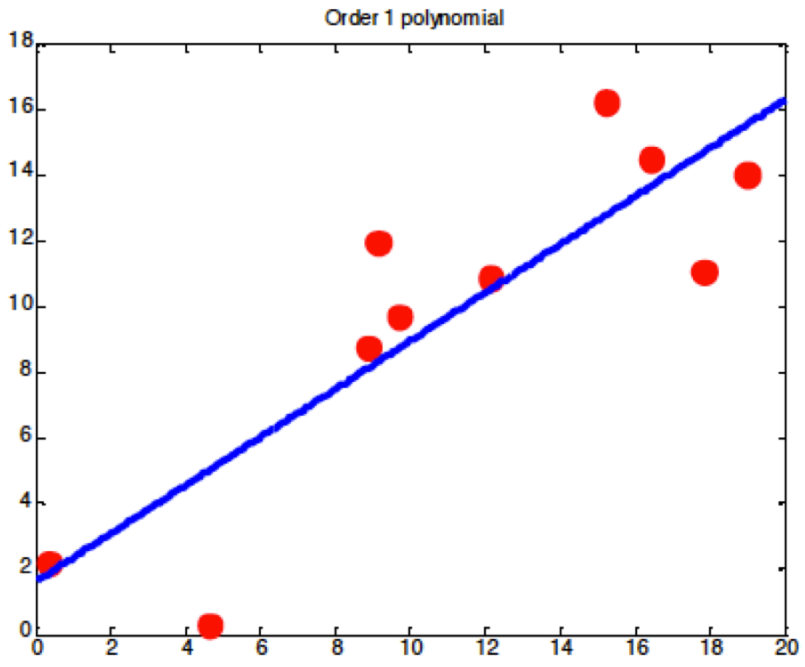
Unsupervised Learning

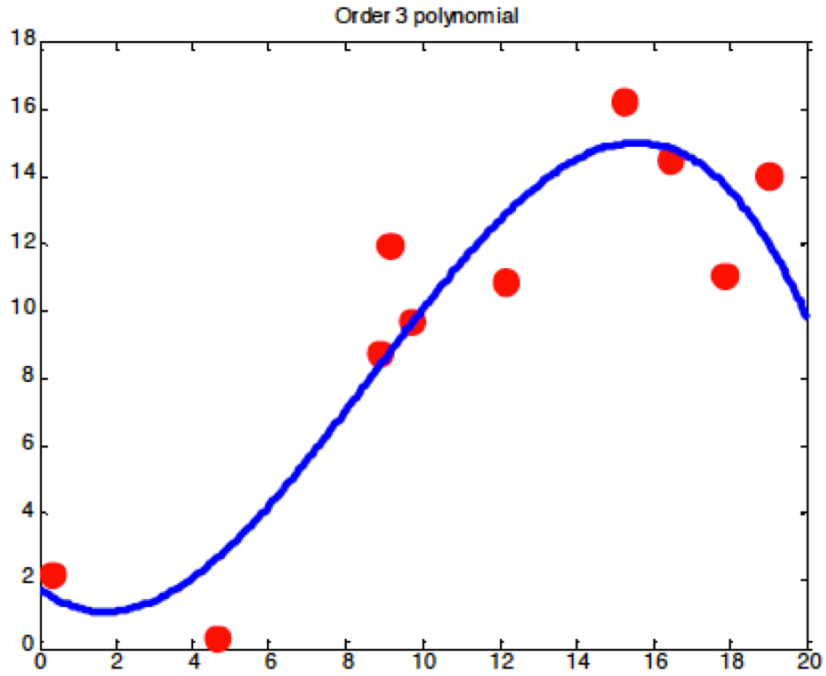
Reinforcement Learning



Transformations of Features

Fitting a line isn't always ideal





(c) Alexander Ihler

Can fit **linear** functions to **nonlinear** features

A nonlinear function of x:

$$\hat{y}(x_i) = \theta_0 + \theta_1 x_i + \theta_2 x_i^2 + \theta_3 x_i^3$$

Can be written as a linear function of $\phi(x_i) = [1 \; x_i \; x_i^2 \; x_i^3]$

$$\hat{y}(x_i) = \sum_{g=1}^{4} \theta_g \phi_g(x_i) = \theta^T \phi(x_i)$$

"Linear regression" means linear in the parameters (weights, biases)

Features can be arbitrary transforms of raw data

What feature transform to use?

- Anything that works for your data!
  - sin / cos for periodic data
  - polynomials for high-order dependencies

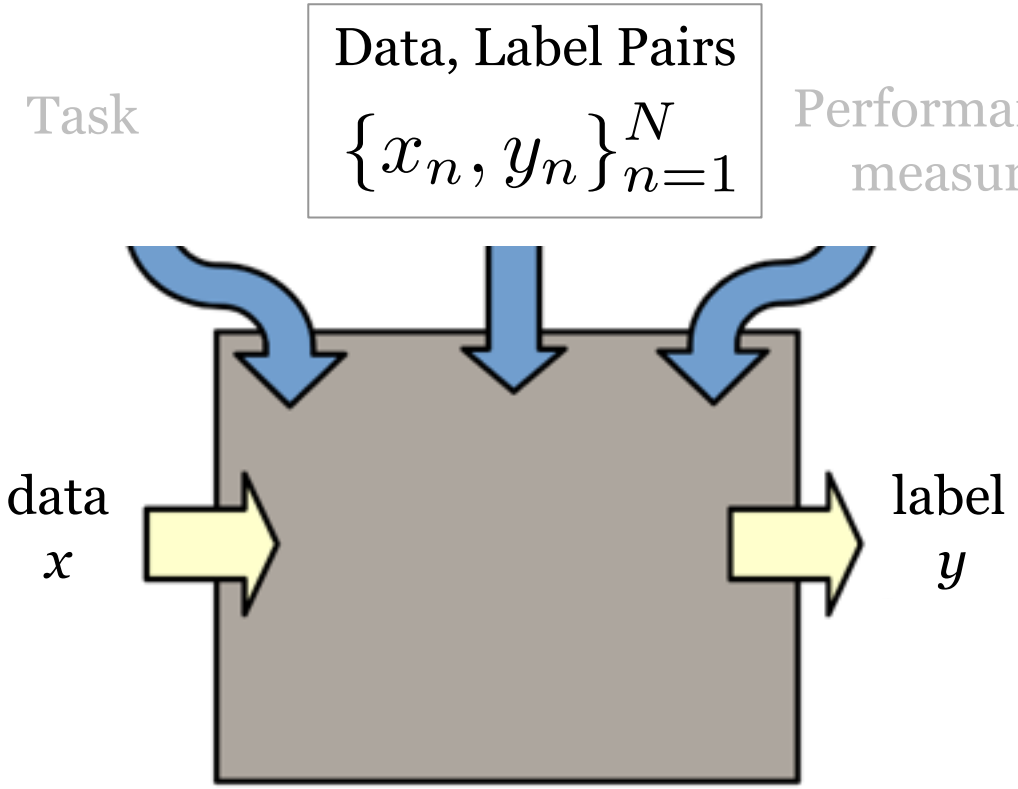
$$\phi(x_i) = [1 \; x_i \; x_i^2 \; \ldots]$$

  - interactions between feature dimensions

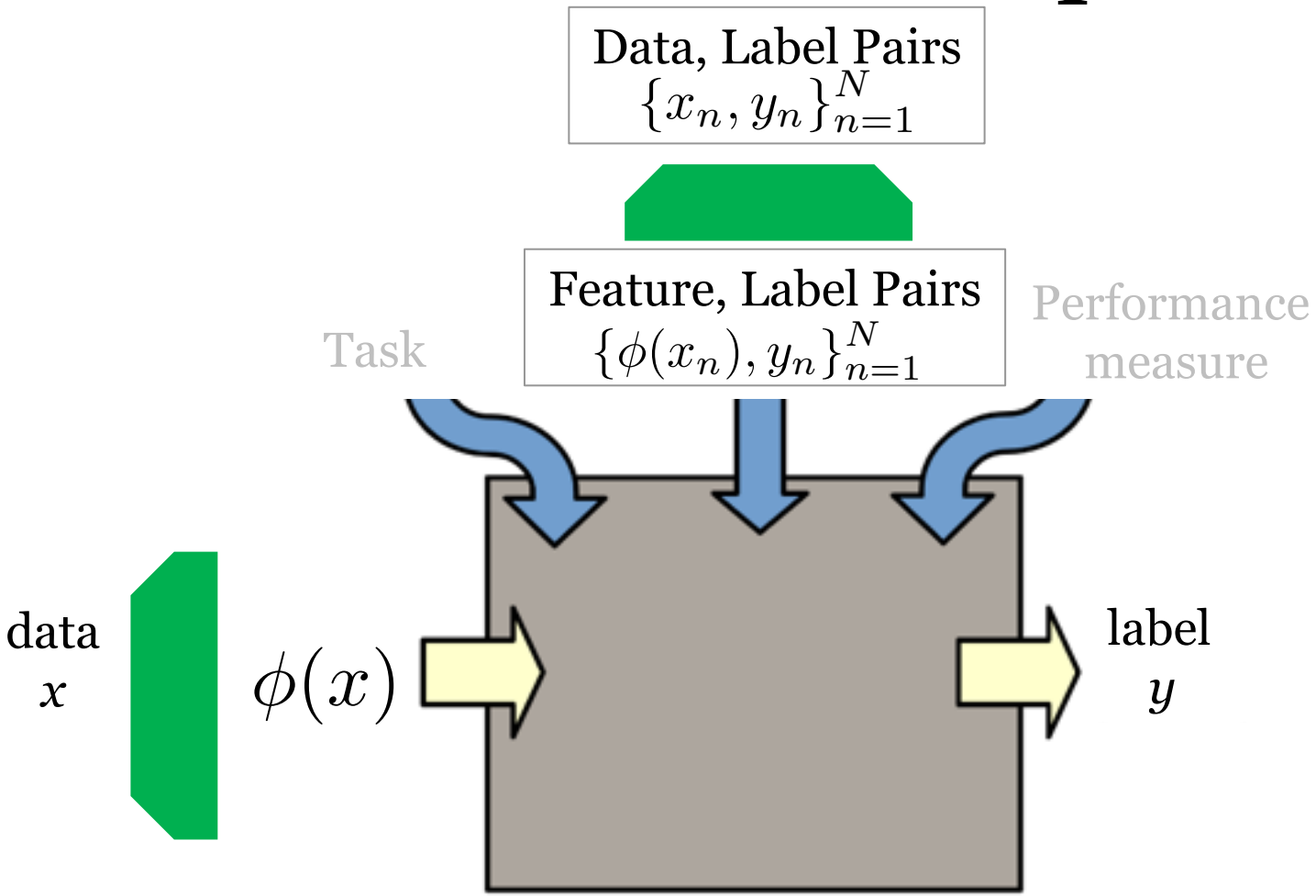
$$\phi(x_i) = [1 \quad x_{i1}x_{i2} \quad x_{i3}x_{i4} \; \ldots]$$

  - Many other choices possible

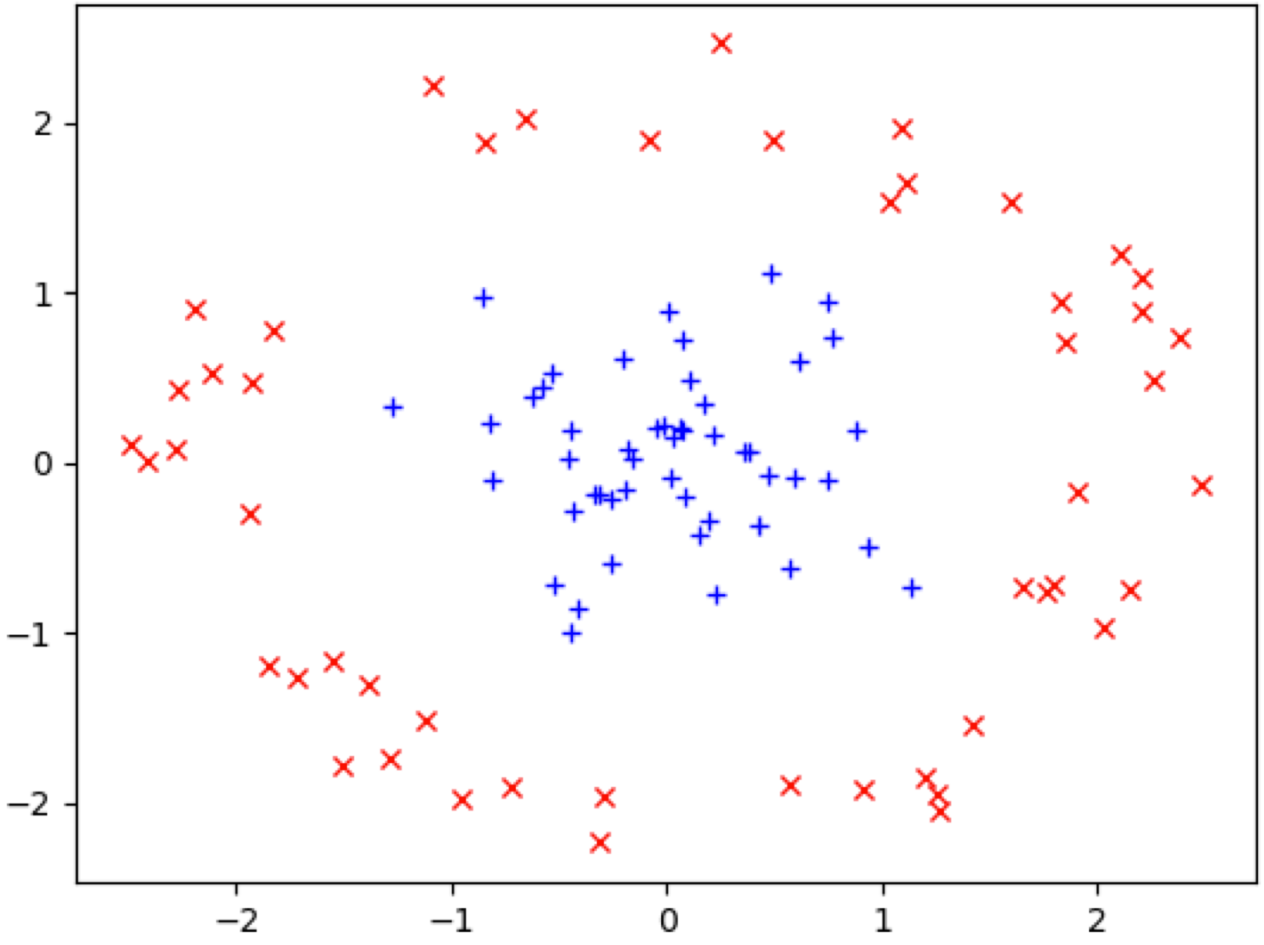
Standard Pipeline



Feature Transform Pipeline



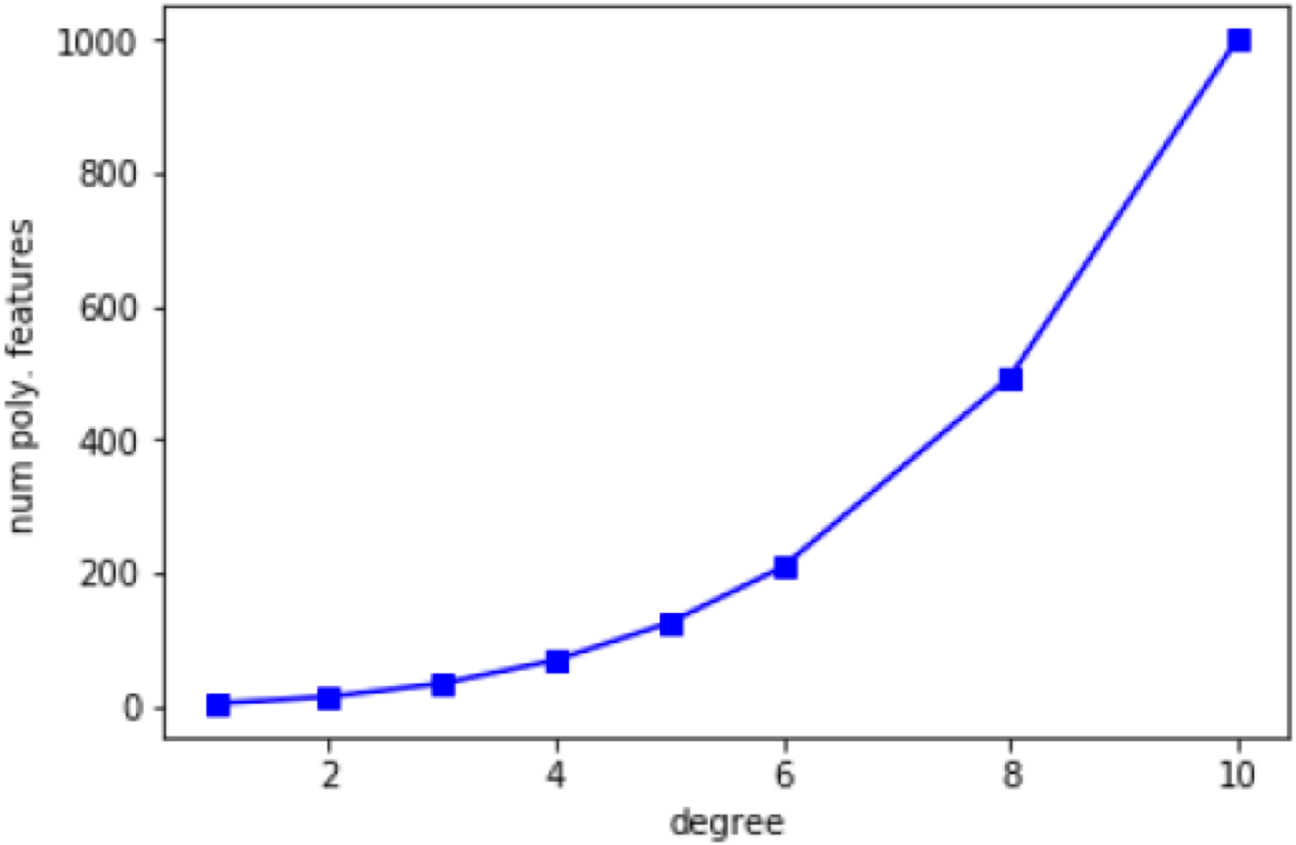
What features to use here?



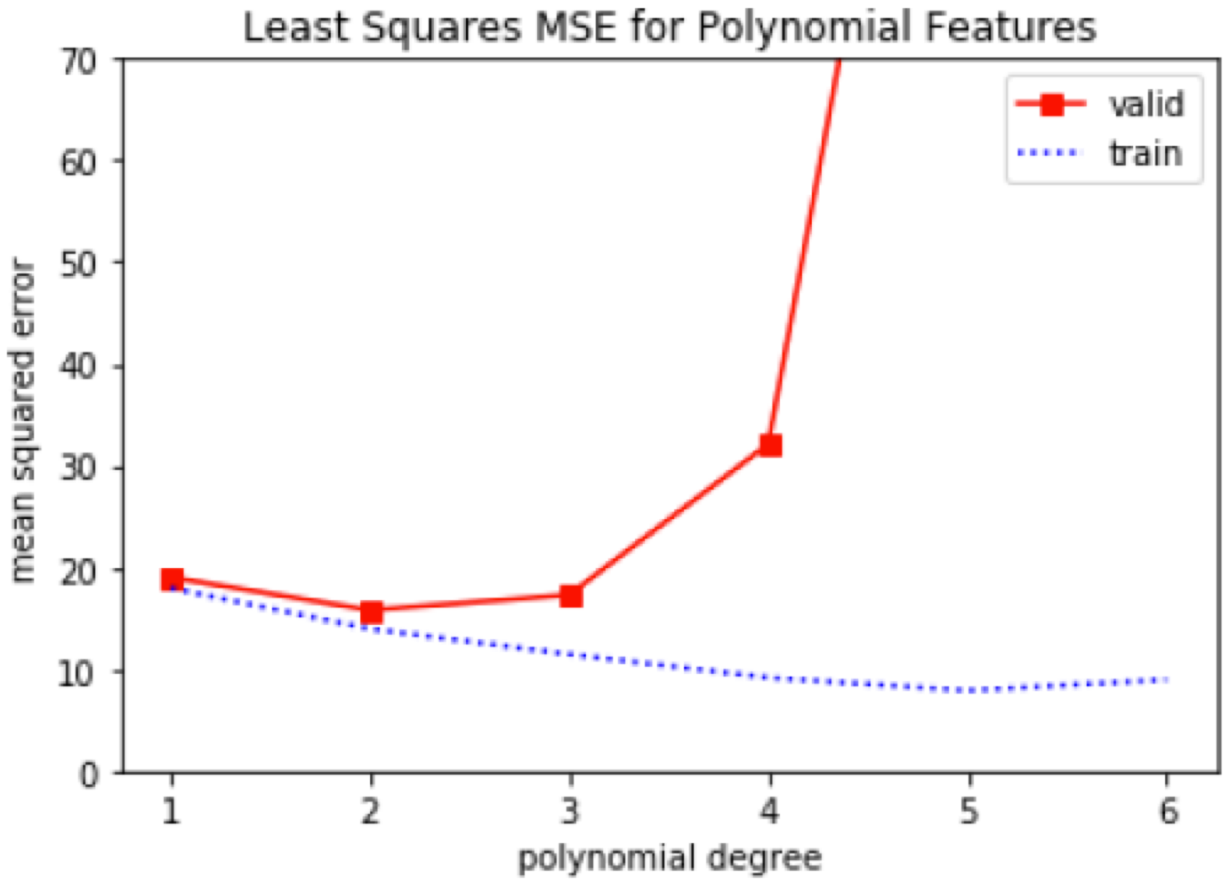
Reasons for Feature Transform

- Improve prediction quality
- Improve interpretability
- Reduce computational costs
  - Fewer features means fewer parameters
- Improve numerical performance of training

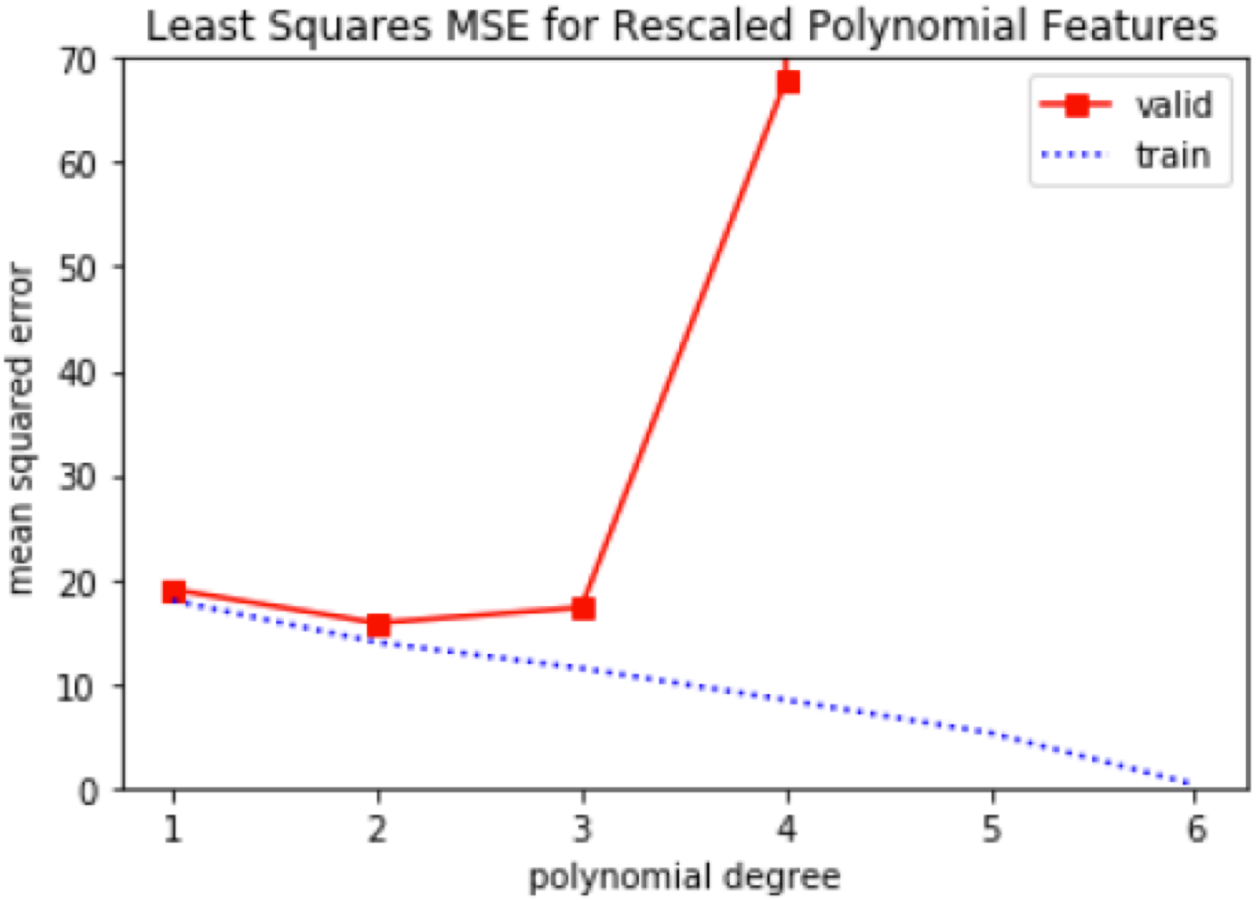
Recall from HW2
Polynomial Features



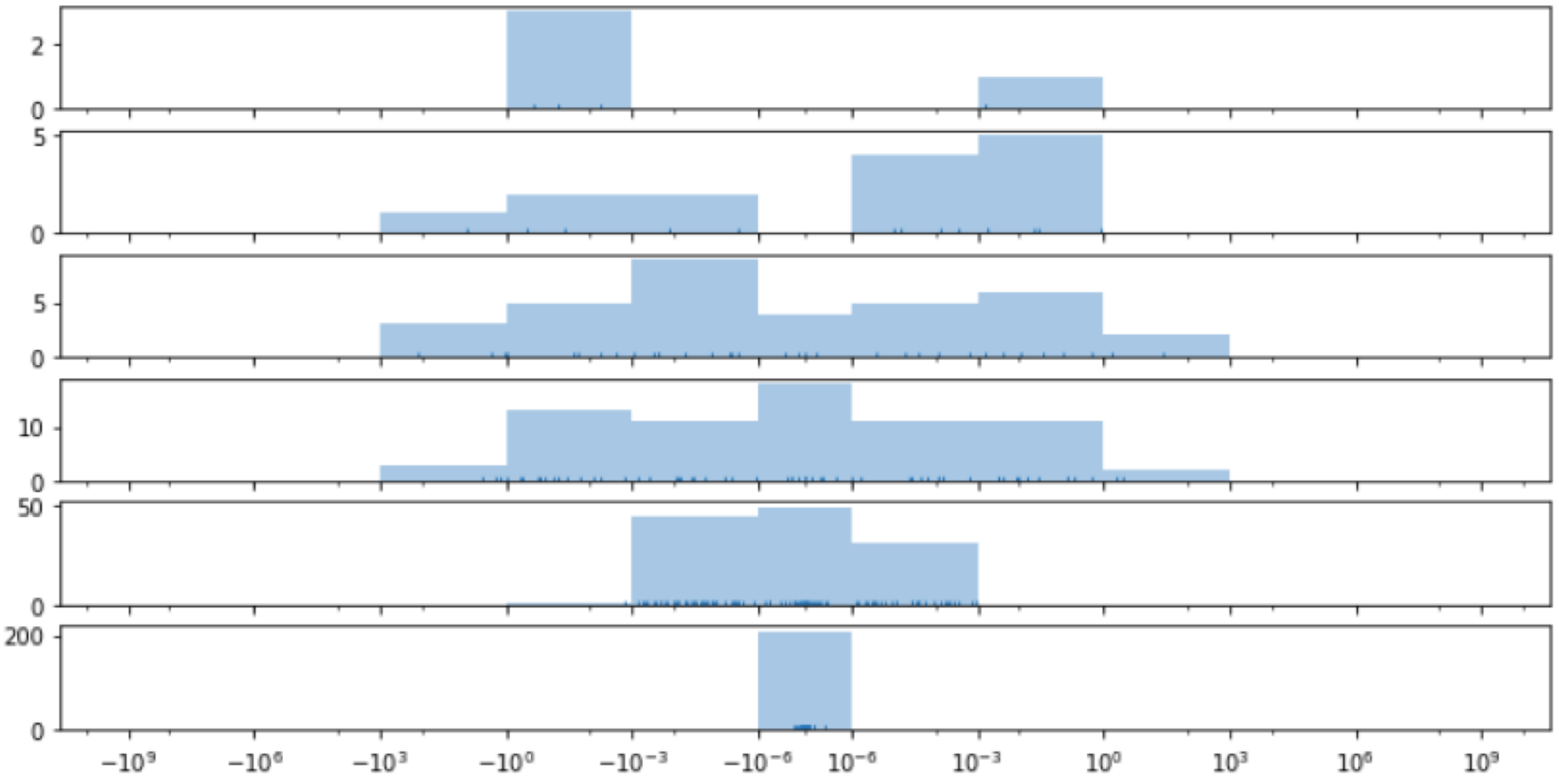
Error vs. Degree (orig. poly.)



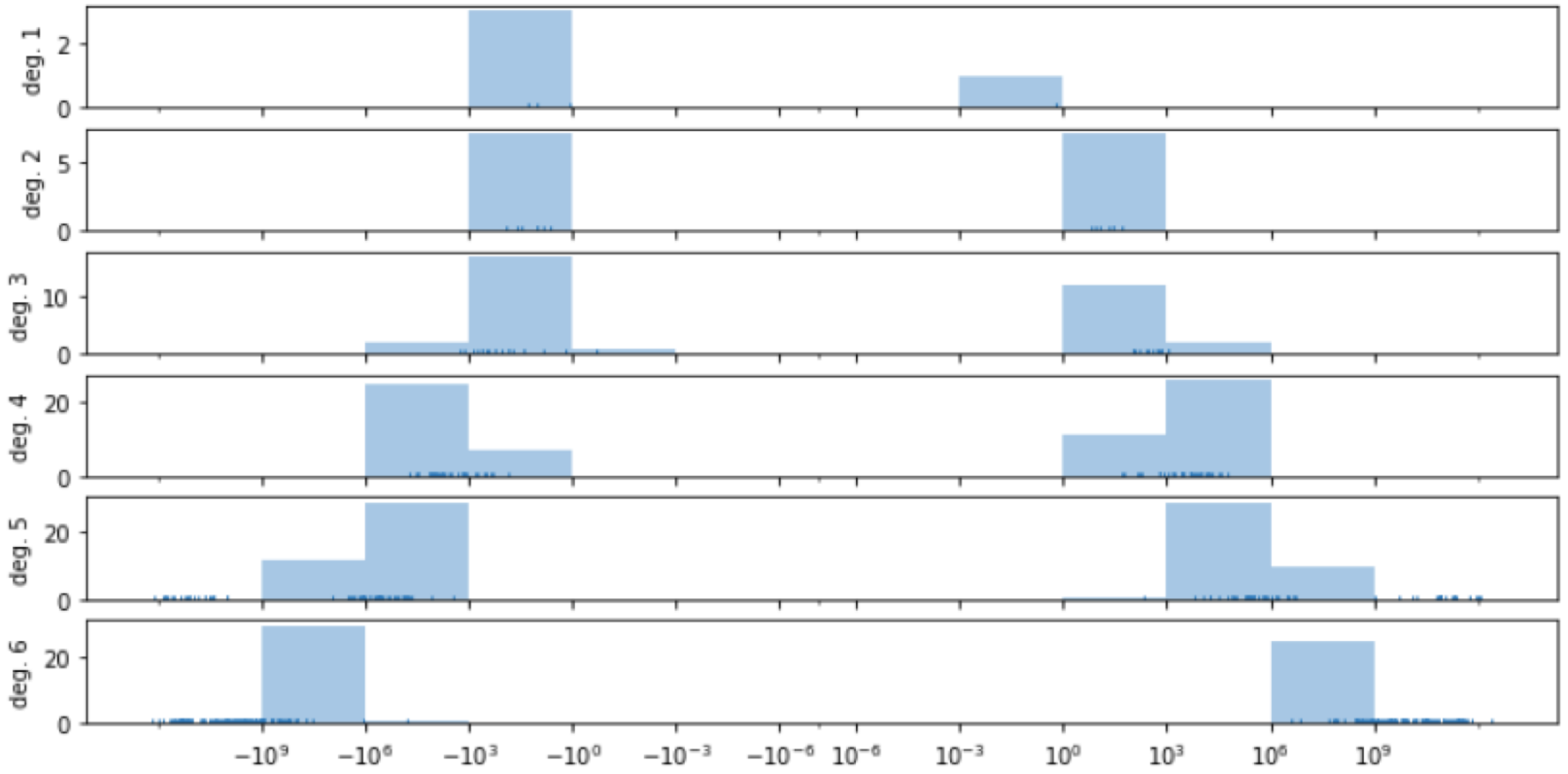
Error vs. Degree (rescaled poly)



Weight histograms (orig. poly.)



Weight histograms (rescaled poly.)



Scikit-Learn Transformer API

```
# Construct a "transformer"
>>> t = Transformer()

# Train any parameters needed
>>> t.fit(x_NF) # y optional, often unused

# Apply to extract new features
>>> feat_NG = t.transform(x_NF)
```

Example 1: Sum of features

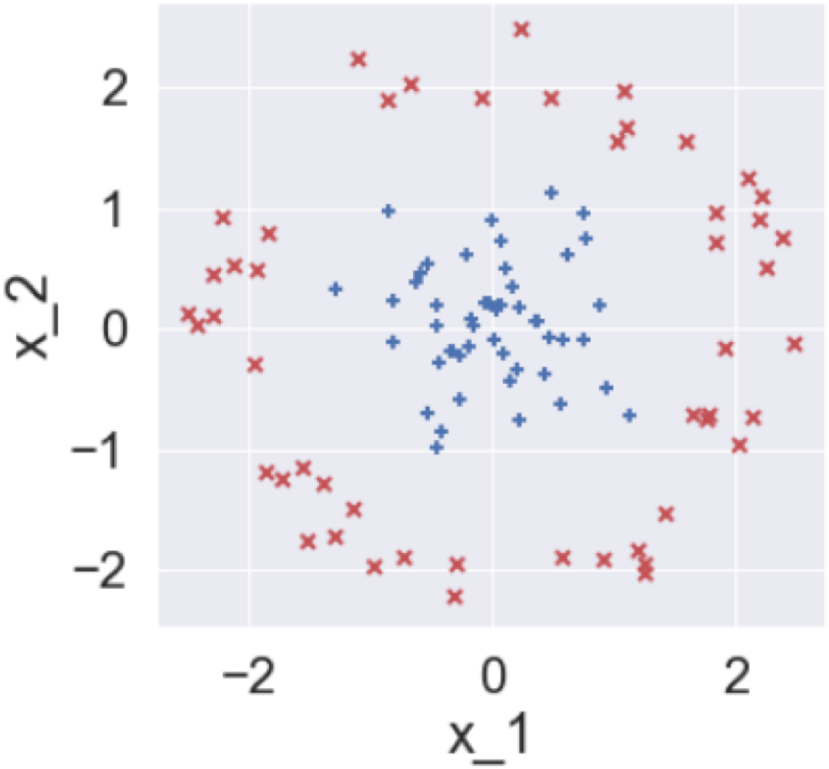
```
From sklearn.base import TransformerMixin
class SumFeatureExtractor(TransformerMixin):
    """ Extracts *sum* of feature vector as new feat
    """
    def __init__(self):
        pass

    def fit(self, x_NF):
        return self

    def transform(self, x_NF):
       return np.sum(x_NF, axis=1)[:,np.newaxis]
```

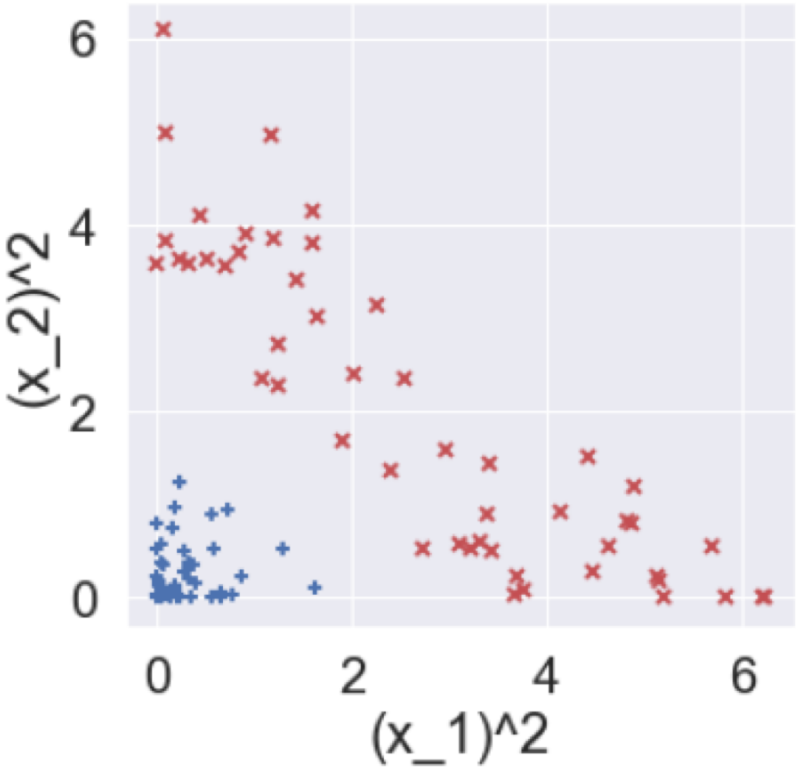
Example 2: Square features

```
From sklearn.base import TransformerMixin
class SquareFeatureExtractor(TransformerMixin):
    """ Extracts *square* of feature vector as new feat
    """
    def fit(self, x_NF):
        return self

    def transform(self, x_NF):
        TODO
```

Example 2: Square features

```
From sklearn.base import TransformerMixin
class SquareFeatureExtractor(TransformerMixin):
    """ Extracts *square* of feature vector as new feat
    """
    def fit(self, x_NF):
        return self

    def transform(self, x_NF):
        return np.square(x_NF)
        # OR return np.power(x_NF, 2)
```


2
1
0
−1
−2
x_2
−2
0
2
x_1


6
4
2
0
(x_2)^2
0
2
4
6
(x_1)^2

Feature Rescaling

Input: Each numeric feature has arbitrary min/max

- Some in [0, 1], Some in [-5, 5], Some [-3333, -2222]

Transformed feature vector

- Set each feature value f to have [0, 1] range

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

- min_f = minimum observed in training set
- max_f = maximum observed in training set

Example 3: Rescaling features

```
From sklearn.base import TransformerMixin
class MinMaxScaleFeatureExtractor(TransformerMixin):
    """ Rescales features between 0 and 1
    """
    def fit(self, x_NF):
        self.min_F = # TODO
        self.max_F = # TODO

    def transform(self, x_NF):
        # TODO
```

Example 3: Rescaling features

```
From sklearn.base import TransformerMixin
class MinMaxFeatureRescaler(TransformerMixin):
    """ Rescales each feature column to be within [0, 1]
        Uses training data min/max
    """
    def fit(self, x_NF):
        self.min_1F = np.min(x_NF, axis=0, keepdims=1)
        self.max_1F = np.max(x_NF, axis=0, keepdims=1)

    def transform(self, x_NF):
        feat_NF = ((x_NF - self.min_1F)
            / (self.max_1F - self.min_1F))
        return feat_NF
```

Feature Standardization

Input: Each feature is numeric, has arbitrary scale

Transformed feature vector

- Set each feature value f to have zero mean, unit variance

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

μ_f Empirical mean observed in training set

σ_f Empirical standard deviation observed in training set

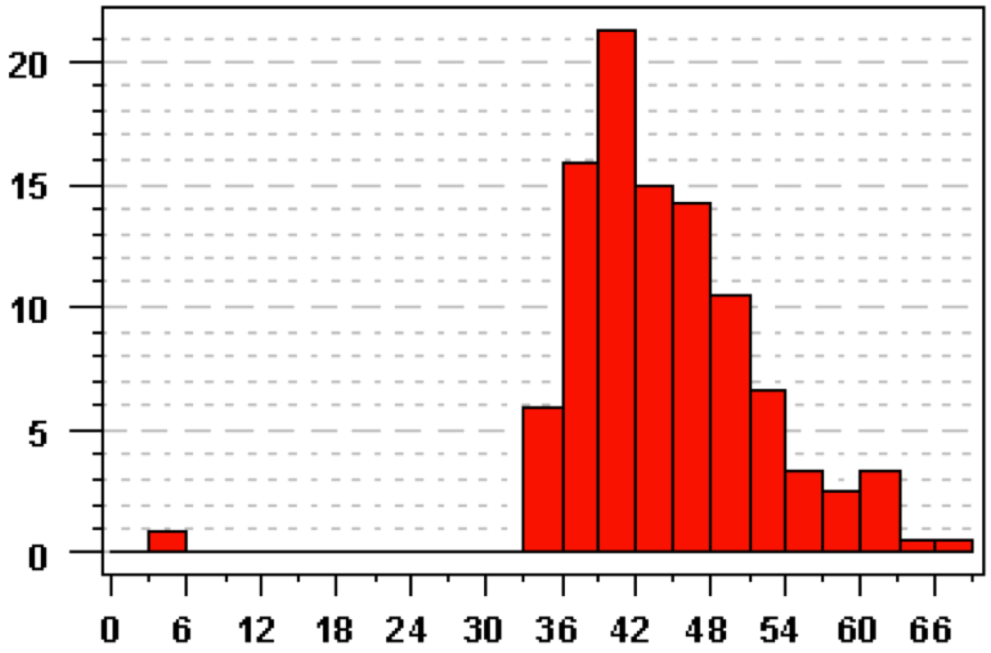
Feature Standardization

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

- Treats each feature as "Normal(0, 1)"
- Typical range will be -3 to +3
  - *If original data is approximately normal*
- Also called z-score transform

Feature Scaling with Outliers

- What happens to standard scaling when training data has outliers?



Feature Scaling with Outliers

sklearn.preprocessing.RobustScaler

class `sklearn.preprocessing.` **`RobustScaler`** (*with_centering=True, with_scaling=True, quantile_range=(25.0, 75.0), copy=True*) [source]

Scale features using statistics that are robust to outliers.

This Scaler removes the median and scales the data according to the quantile range (defaults to IQR: Interquartile Range). The IQR is the range between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile).

Combining several transformers

```
from sklearn.pipeline import FeatureUnion
```

```
union_transformer = FeatureUnion(transformer_list=[
    ('sum_x', SumFeatureExtractor()),
    ('square_x', SquareFeatureExtractor()),
    ('rescale_x', MinMaxFeatureRescaler()),
])
```

```
union_transformer.fit(x_N2);
union_transformer.transform(x_N2)[:3]
```

```
array([[-2.19,  5.19,  0.01,  0.04,  0.49],
       [-3.04,  3.41,  1.43,  0.13,  0.22],
       [ 1.81,  0.01,  3.59,  0.48,  0.88]])
```

Categorical Features

```
["uses Firefox", "uses Chrome", "uses
Safari", "uses Internet Explorer"]
```

Numerical encoding

"uses Firefox" → 1

"uses Safari" → 3

Categorical Features

```
["uses Firefox", "uses Chrome", "uses
Safari", "uses Internet Explorer"]
```

One-hot vector

	Firefox	Chrome	Safari	Internet Explorer
"uses Firefox"	1	0	0	0
"uses Safari"	0	0	1	0

Feature Selection or “Pruning”

Best Subset Selection

Algorithm 6.1 *Best subset selection*

1. Let $\mathcal{M}_0$ denote the *null model*, which contains no predictors. This model simply predicts the sample mean for each observation.
2. For $k = 1, 2, \ldots p$:
   (a) Fit all $\binom{p}{k}$ models that contain exactly k predictors.
   (b) Pick the best among these $\binom{p}{k}$ models, and call it $\mathcal{M}_k$. Here *best* is defined as having the smallest RSS, or equivalently largest R^2.
3. Select a single best model from among $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.

Problem: Too many subsets!

there are 2^p models that involve subsets of p predictors. So if $p = 10$, then there are approximately 1,000 possible models to be considered, and if $p = 20$, then there are over one million possibilities!

Forward Stepwise Selection

Start with zero feature model (guess mean)
Store as M_0

Add best scoring single feature (search among F)
Store as M_1

For each size k = 2, … F
    Try each possible not-included feature (F – k + 1)
    Add best scoring feature to the model M_k-1
    Store as M_k

Pick best among M_0, M_1, … M_F on **validation**

Best vs Forward Stepwise

# Variables	Best subset	Forward stepwise
One	`rating`	`rating`
Two	`rating`, `income`	`rating`, `income`
Three	`rating`, `income`, `student`	`rating`, `income`, `student`
Four	`cards`, `income`, `student`, `limit`	`rating`, `income`, `student`, `limit`

TABLE 6.1. *The first four selected models for best subset selection and forward stepwise selection on the* `Credit` *data set. The first three models are identical but the fourth models differ.*

Easy to find cases where forward stepwise 's greedy approach doesn't deliver best possible subset.

Backwards Stepwise Selection

Start with all features

Gradually test all models with one feature removed.

Repeat.

Other Feature Selection Methods

- Remove features with low variance

- Select to maximize mutual information

Missing Data: Imputation

- https://scikit-learn.org/stable/modules/impute.html#impute

Properties of Good Features

- Informative
- Independent
- Monotonic with predictive probability
  - If monotonic, linear decision boundaries possible