Regularisation

Overview

- Regularization
- Unsupervised learning
    - PCA decomposition
    - K-means clustering

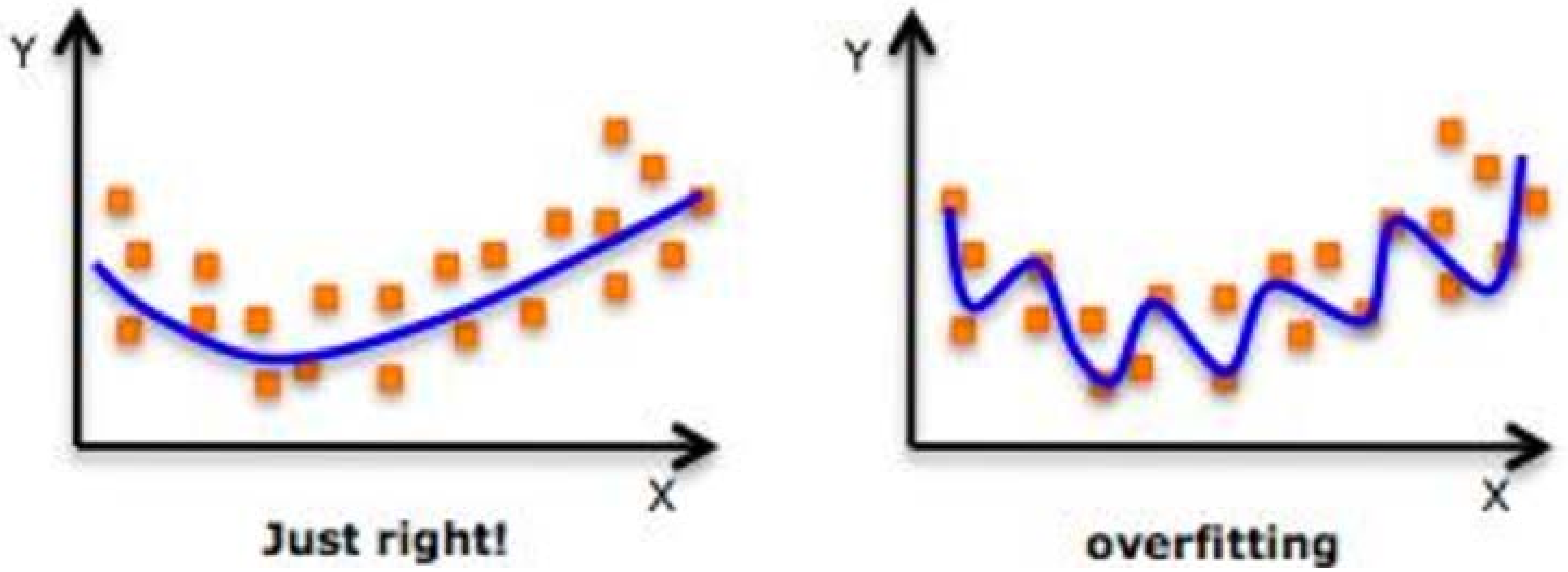
Keep it simple

Rule of thumb: *always choose the simplest model among those with the same test error*

- Linear models
  - Number of non-zero weights in the model
- Tree-based models
  - Number of splits or depth

Reasons:

- Over-fitting
  - Complex models can explain training data too well



- Maintenance
  - Easier to interpret
  - Less memory, faster predictions

Some ideas

Features: A, B, C, … , I
Target: Y = {0, 1}

- Restrict a set of features

- Remove features that Y doesn't depend on
  - There are statistical tests to measure X ~ Y (chi2, F-test)
  - It's wrong to compare results of different X_i~Y tests

- Change the learning process

- Logistic regression:

$$L(w) = \sum_{i=1}^{n} y_i \ln p_i + (1 - y_i) \ln(1 - p_i) \rightarrow min$$

$$p_i(w) = Logit(w_0 + w_1 * A_i + w_2 * B_i + \ldots w_m * I_i)$$

- Lets introduce penalty R(w)

$$L^{'}(w) = L(w) + R(w) \rightarrow min$$

- Suboptimal on training sample is OK

- L2 penalty

$$R(w) := \lambda \sum_{j=1}^{m} w_j^2 = \lambda ||\vec{w}||^2$$

- L'(w) is smooth
  - 1st and 2nd order methods work fine
- Having large weights is expensive now

Outcome

- Helps to fight over-fitting
  - Worse error on training sample
  - Possibly/usually better on test

- Doesn't make model more sparse
  - Though weights close to 0

- L1 regularization: $$R(w) := \lambda \sum_{j=1}^{m} |w_j|$$
  - No derivative at 0

- Need to adapt optimization algorithm!
  - Proximal operator is used in Spark.mllib

- Result – sparse solution (many exact 0)

- Elastic net is a generalization

$$R(w) := \alpha\lambda \sum_{j=1}^{m} |w_j| + (1-\alpha)\frac{\lambda}{2}\sum_{j=1}^{m} w_j^2$$

  - L2: Alpha = 0
  - L1: Alpha = 1
  - Now two parameters

- Keep data reasonably scaled
  - X_1 in [0, 100]; X_2 in [0, 1]
  - Changes in w1 and w2 have different effect
  - Both penalized in the same way
- Rescaling ideas
  - Standardization – divide by variance, optionally exclude mean
  - Scale to [0, 1] using min/max values
- Lambda/Alpha are hyperparameters
  - Tune it!

```
lr = LinearRegression(elasticNetParam=0, regParam=0)
model = lr.fit(train)
```

```
test1= model.transform(test)
evaluator.evaluate(testl, {evaluator.metricName:"r2"})
```

```
0.7513452384843318
```

```
model.coefficients.numNonzeros()
```

```
10
```

```
print model.coefficients.array
```

```
[    503.64037616    1978.78195227     -34.08383187    -401.6181249
      93.08924586    -628.08538433   -4092.57943636   10476.81327403
    -883.17338048   -2013.57664089]
```

```
lr = LinearRegression(elasticNetParam=0, regParam=100)
model = lr.fit(train)
```

```
test1= model.transform(test)
evaluator.evaluate(test1, {evaluator.metricName:"r2"})
```

```
0.7702004968536114
```

```
model.coefficients.numNonzeros()
```

```
10
```

```
print model.coefficients.array
```

```
[  444.18197632  1892.95220966   -14.32696612  -427.46250048    87.57063636
  -612.65452573  2155.54927175  3322.79712774  -827.31108065 -2278.14286611]
```

```
lr = LinearRegression(elasticNetParam=1, regParam=50)
model = lr.fit(train)
```

```
test1= model.transform(test)
evaluator.evaluate(testl, {evaluator.metricName:"r2"})
```

```
0.757677565768841
```

```
model.coefficients.numNonzeros()
```

```
8
```

```
print model.coefficients.array
```

```
[  385.34592239  1904.6600738      0.            -90.76236387    71.09064981
  -615.87367552     0.          5701.65441192  -448.06856749 -1503.82530631]
```

Recap

- L1/L2 regularization is a good way to improve generalization
- L1 provides sparsity (feature selection)
- This idea goes beyond linear models
  - Neural networks
  - Support Vector Machines