House Prices Example

Sanjay Lall and Stephen Boyd

EE104
Stanford University
The data set

- sale prices of $n = 1456$ homes in Ames, Iowa from 2006 to 2010
- goal is to predict $\log(\text{price})$
- performance metric is RMS error on test set
- e.g., RMS error of 0.1 means (roughly) we can predict house price within factor $e^{0.1}$ (about $\pm 10.5\%$)
Scatter plot of price versus living area

$\log(\text{price})$ vs. living area
Embedding

- $v =$ price, let $y = \log(v)$

- 9 numerical fields are embedded unchanged
  - year built, area of living space, area of first floor, area of second floor, area of garage, area of wooden deck, area of basement, year of last remodel, area of lot

- 8 ordinal fields are embedded as integers
  - number of bedrooms, number of kitchens, number of fireplaces, number of half bathrooms, number of rooms, condition (scored 1-10), quality of materials and finish (scored 1-10), car capacity of garage
Embedding

- **kitchen quality**: on Likert scale
  
  EXCELLENT, GOOD, TYPICAL, FAIR, POOR

  embedded as integer between 1 and 5

- **building type**: 5 categories, one-hot embedded
  
  SINGLE-FAMILY    TOWNHOUSE END UNIT    TWO-FAMILY-CONVERSION
  TOWNHOUSE INSIDE UNIT    DUPLEX

- **neighborhood**: 25 categories, one-hot embedded

- results in matrix $X_0 \in \mathbb{R}^{n \times 48}$
Standardization and data splitting

- split randomly 80/20 into training and test sets
- gives $X_{\text{train}} \in \mathbb{R}^{1165 \times 48}$, $Y_{\text{train}} \in \mathbb{R}^{1165}$ and $X_{\text{test}} \in \mathbb{R}^{291 \times 48}$ and $Y_{\text{test}} \in \mathbb{R}^{291}$
- use training set to compute means and standard deviations of each column of $X_{\text{train}}$
- use means and stds to standardize $X_{\text{train}}$ and $X_{\text{test}}$
  - both datasets are standardized using the same means/stds (from the training set)
  - some columns of $X_{\text{train}}$ may have zero standard deviation (e.g. categoricals which occur rarely); special case since standardization formula does not apply
- append a constant feature; let $X_{\text{train}} = \begin{bmatrix} 1 & \text{standardize}(X_{\text{train}}) \end{bmatrix}$ and $X_{\text{test}} = \begin{bmatrix} 1 & \text{standardize}(X_{\text{test}}) \end{bmatrix}$
Ridge regression

- choose a range of $\lambda$ values, logarithmically spaced between $10^{-3}$ and $10^3$
- for each $\lambda$, compute
  - RERM: the $\theta$ that minimizes $\frac{1}{n} \| X^{\text{train}} \theta - Y^{\text{train}} \|^2 + \lambda \| \theta_{2:d} \|^2$
  - the training error $\text{rms}(X^{\text{train}} \theta - Y^{\text{train}})$
  - the test error $\text{rms}(X^{\text{test}} \theta - Y^{\text{test}})$
Ridge regression

- no benefit of regularization in this case
- minimum rms error on test set is about 0.12
- corresponds to about 13% error in house price
Results

- plot shows all test points
Important features

- $\theta_2$: year built
- $\theta_3$: area of living space
- $\theta_4$: area of first floor
- $\theta_5$: area of second floor
- $\theta_8$: area of basement
- $\theta_{16}$: condition
- $\theta_{17}$: quality of materials and finish
- Difference between best and worst neighborhoods is 4% price
D, header = loaddata()
n = size(D,1)
Y = embedy(D, header)
X0 = embedx(D, header)
trainrows, testrows = randomsplit(n)
Xtrain0, Ytrain, Xtest0, Ytest = applysplit(X0, Y, trainrows, testrows)
means, stds = getstatistics(Xtrain0)
Xtrain = standardizeplusone(Xtrain0, means, stds)
Xtest = standardizeplusone(Xtest0, means, stds)
lambdas = 10.^range(-3,3,length=50)
thetas = [ridgeregressionconstfeature(Xtrain, Ytrain, lambda) for lambda in lambdas]
train_errors = [rmse(Xtrain*theta, Ytrain) for theta in thetas]
test_errors = [rmse(Xtest*theta, Ytest) for theta in thetas]
function randomsplit(n, trainfrac=0.8)
    ntrain = convert(Int64, round(trainfrac*n))
    p = Random.randperm(n)
    trainrows = sort(p[1:ntrain])
    testrows = sort(p[ntrain+1:n])
    return trainrows, testrows
end

function applysplit(X, Y, trainrows, testrows)
    return X[trainrows,:], Y[trainrows,:], X[testrows,:], Y[testrows,:]
end
function getstatistics(U)
    means = [Statistics.mean(x) for x in eachcol(U)]
    stds = [Statistics.std(x) for x in eachcol(U)]
    return means, stds
end
function standardizeplusone(X,means,stds)
    Z = zeros(size(X))
    for i=1:size(X,2)
        if stds[i] != 0
            Z[:,i] = (X[:,i] .- means[i])/stds[i]
        else
            Z[:,i] = X[:,i] .- means[i]
        end
    end
    n = size(X,1)
    Z = [ones(n,1) Z]
    return Z
end
```julia
function embedx(D, header)
    field(name) = getdatafield(D, header, name)
    realf(name) = stringtonumber.(field(name))
    X = hcat(realf("YearBuilt"), # numeric
             realf("GrLivArea"), # numeric
             realf("1stFlrSF"), # numeric
             ...
             realf("GarageCars"), # ordinal 0-4
             unlikert.(field("KitchenQual")), # ordinal, but "Ex", "Gd", "TA", "Fa", "Po"
             onehot(field("Neighborhood")), # 25 different names
             onehot(field("BldgType")), # 5 different types
            )
    return X
end
```
function unlikert(s)
    d = Dict("Ex" => 5, "Gd" => 4, "TA" => 3, "Fa" => 2, "Po" => 1)
    return d[s]
end

embedy(D, header) = log.(stringtonumber.(getdatafield(D, header, "SalePrice")))
# takes a list length n, e.g. u = ["hi", "lo", "hi", "med", "lo"] # returns a matrix Y which is n by d

```julia
function onehot(u)
    categories = unique(u)
    catnum(s) = findfirst(x -> x==s, categories)
    n = length(u)
    K = length(categories)
    Y = zeros(n,K)
    for i=1:n
        c = catnum(u[i])
        Y[i,c] = 1
    end
    return Y
end
```
function ridgeregressionconstfeature(X,Y,lambda)
    n,d = size(X)
    m = size(Y,2)
    E = [zeros(d-1,1) I(d-1)]
    A = [X; sqrt(lambda*n)*E]
    B = [Y; zeros(d-1,m)]
    theta = A\B
end