# Unit 4
# Feature Selection and LASSO

Prof. Phil Schniter

THE OHIO STATE UNIVERSITY

ECE 4300: Introduction to Machine Learning, Sp20

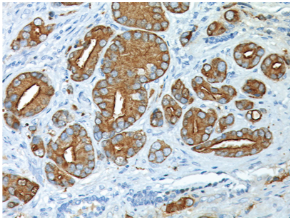
## Learning objectives

- Understand motivation and idea behind feature selection
- Understand feature selection methods based on:
  - exhaustive search
  - stepwise selection
  - target cross-correlation
  - regularization
- Understand ridge regression and LASSO:
  - interpret their coefficient paths
  - implement LASSO using sklearn
  - know how to select the regularization strength using cross-validation
- Understand connections to ML estimation and MAP estimation

## Outline

## Prostate-specific antigen (PSA) testing

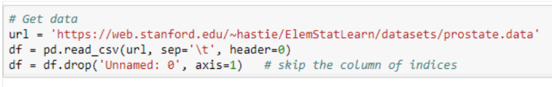
- High PSA is linked to prostate cancer
  - PSA levels are easily monitored
  - A common tool for screening
- Classic 1989 study by Thomas et al:
  - Measured PSA level of 102 men prior to prostate removal
  - Measured various biometrics
  - Biometrics include cancer volume, prostate weight, age, etc.
- Machine-learning problem:
  - Can we predict PSA from these biometrics?



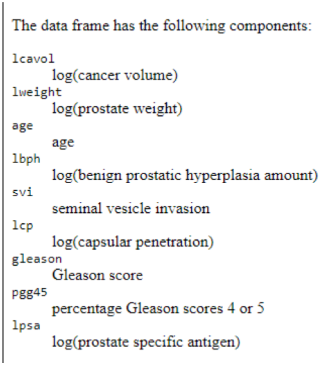
**Stamey, et al.**, "Prostate specific antigen in the diagnosis and treatment of adenocarcinoma of the prostate. II. Radical prostatectomy treated patients," ***The Journal of Urology***, **141.5 (1989): 1076-1083.**

## PSA Dataset

- Prostate dataset widely used in ML classes
- Can be downloaded from many websites
- Data samples from 97 patients
- 8 features, shown on right
- Target variable = lpsa (log PSA)



```
# Get data
url = 'https://web.stanford.edu/~hastie/ElemStatLearn/datasets/prostate.data'
df = pd.read_csv(url, sep='\t', header=0)
df = df.drop('Unnamed: 0', axis=1)   # skip the column of indices
```



The data frame has the following components:

| | |
|---|---|
| `lcavol` | log(cancer volume) |
| `lweight` | log(prostate weight) |
| `age` | age |
| `lbph` | log(benign prostatic hyperplasia amount) |
| `svi` | seminal vesicle invasion |
| `lcp` | log(capsular penetration) |
| `gleason` | Gleason score |
| `pgg45` | percentage Gleason scores 4 or 5 |
| `lpsa` | log(prostate specific antigen) |

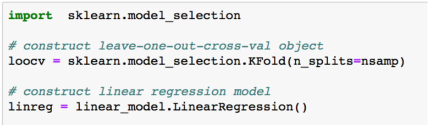
## First attempt: Linear Regression

- Let's try first with multiple linear regression:

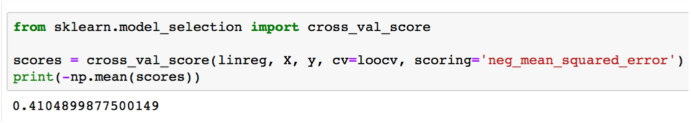
$$y \approx \widehat{y} = \beta_0 + \beta_1 x_1 + \cdots + \beta_d x_d$$

  - $y =$ lpsa (target PSA level)
  - $\{x_j\}_{j=1}^d$ are biometric features with $d = 8$

- Why linear regression?
  - Coefficients are easy to fit (via LS)
  - Coefficients are easy to interpret
    - larger $|\beta_j|$ means $x_j$ has larger effect on PSA



```
import  sklearn.model_selection

# construct leave-one-out-cross-val object
loocv = sklearn.model_selection.KFold(n_splits=nsamp)

# construct linear regression model
linreg = linear_model.LinearRegression()
```



```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(linreg, X, y, cv=loocv, scoring='neg_mean_squared_error')
print(-np.mean(scores))
```

```
0.4104899877500149
```

- Can we do better?

## Outline

## Feature Selection

- From last lecture:
  - Too many features $\Rightarrow$ large error variance
  - This motivates using fewer features. But which ones?
  - Feature selection: use only the best *subset* of $d$ total features
- Feature selection via exhaustive search:
  - Main idea: use K-fold CV to test *every possible* subset
  - This is the optimal approach to feature selection
  - But, with large $d$, testing $2^d$ subsets may be computationally impractical!
- Suboptimal feature selection methods:
  - Stepwise selection
  - Correlation-based methods
  - Regularization-based methods, e.g., LASSO

## Stepwise selection (or stepwise regression)

- Forward selection
  - First use CV to find the single feature yielding the lowest RSS
  - Then, add one of the remaining features so that the pair provides the lowest RSS
  - Repeat until RSS starts to increase, or maximum allowed # features is reached
- Backwards elimination
  - First use all $d$ features and compute the RSS using CV
  - Then, remove one of the features so that the remaining features give lowest RSS
  - Repeat until RSS starts to increase, or only one feature remains
- These methods are suboptimal
  - They look only one step ahead (i.e., "greedy")
  - A better approach is LASSO, discussed later

## Feature selection via cross-correlation

Here is another heuristic strategy. Again, assume $d$ total features.

- Maximize cross-correlation with target:
  - For each $p \in \{1, \dots, d\}$, find the $p$ features that are most correlated with the target (i.e., compute $r_{yx_j} = \frac{1}{n}\sum_{i=1}^{n} y_i x_{ij}$ for each $j$, and choose the $p$ values of $j$ giving largest $|r_{yx_j}|$)
  - Use cross-validation to optimize the model-order $p$
- Is this a good idea?
  - Not necessarily
  - Two features might both be highly correlated with the target, but provide redundant information, in which case only one of them should be used
  - There exist more sophisticated versions that penalize correlations among features https://en.wikipedia.org/wiki/Feature_selection#Correlation_feature_selection
- A better approach is LASSO, discussed next

## Outline

## Regularization

- Previously, we optimized the linear regression coefficients $\boldsymbol{\beta}$ via LS:

$$\widehat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \mathrm{RSS}(\boldsymbol{\beta}) \quad \text{for} \quad \mathrm{RSS}(\boldsymbol{\beta}) = \sum_{i=1}^{n} (y_i - \widehat{y}_i(\boldsymbol{\beta}))^2$$

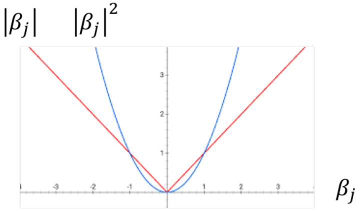
- Can we modify this to perform feature selection?
- Idea: Penalize the use of each feature (i.e., penalize $\beta_j \neq 0$)
  - If the penalty results in $\widehat{\beta}_j = 0$, then the $j$th feature is not used
  - In particular, add a "regularization term" $\phi(\boldsymbol{\beta})$ to the optimization objective:

$$\widehat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} J(\boldsymbol{\beta}) \quad \text{for} \quad J(\boldsymbol{\beta}) = \mathrm{RSS}(\boldsymbol{\beta}) + \phi(\boldsymbol{\beta})$$

    for some $\phi(\cdot)$ that encourages $\widehat{\beta}_j = 0$ for non-informative features $j$
  - How should we choose $\phi(\cdot)$?

## L1 and L2 regularization

- L2 regularization: $\boxed{\phi(\boldsymbol{\beta}) = \alpha \sum_{j=1}^{d} |\beta_j|^2}$
- L1 regularization: $\boxed{\phi(\boldsymbol{\beta}) = \alpha \sum_{j=1}^{d} |\beta_j|}$
- Both penalize $\beta_j \neq 0$, but in different ways
- The overall strength of regularization is controlled by $\alpha \geq 0$
- Note: regularization does *not* involve the intercept term $\beta_0$, since we do *not* want to penalize it!



## Data standardization

- Motivation:
  - The L1 and L2 regularizers penalize all coefficients $\beta_j$ *uniformly*
  - But if some $x_j$ are much bigger than others, then some $\beta_j$ may be much bigger than others, in which case $\beta_j$ should not be treated uniformly
  - Can avoid this issue by normalizing the sizes (i.e., sample variances) of $x_j$
  - If we also remove the mean of $y$ and each $x_j$, then conveniently we get $\beta_0 = 0$
- Procedure:
  - "Standardize" target and features to have sample mean $0$ and sample variance $1$:

$$x_{ij} \leftarrow (x_{ij} - \overline{x}_j)/s_{x_j} \quad \text{and} \quad y_i \leftarrow (y_i - \overline{y})/s_y$$

  - Design a predictor $\boldsymbol{\beta}$ using the standardized data
  - When test data $\boldsymbol{x}$ arrives, either standardize $\boldsymbol{x}$ using the above quantities, or un-standardize the predictor $\boldsymbol{\beta}$ (see homework)

## Ridge regression and LASSO

Assuming $(\boldsymbol{y}, \boldsymbol{X})$ has been standardized, and setting $\boldsymbol{\beta} = [\beta_1, \dots, \beta_d]^\mathsf{T}$,

- Ridge regression cost function:

$$J_{\text{ridge}}(\boldsymbol{\beta}) = \underbrace{\sum_{i=1}^{n} (y_i - \widehat{y}(\boldsymbol{\beta}))^2}_{=\text{RSS}(\boldsymbol{\beta})} + \alpha \underbrace{\sum_{j=1}^{d} |\beta_j|^2}_{\text{L2 regularization}} = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2 + \alpha \|\boldsymbol{\beta}\|^2$$

- LASSO cost function:

$$J_{\text{lasso}}(\boldsymbol{\beta}) = \underbrace{\sum_{i=1}^{n} (y_i - \widehat{y}(\boldsymbol{\beta}))^2}_{=\text{RSS}(\boldsymbol{\beta})} + \alpha \underbrace{\sum_{j=1}^{d} |\beta_j|}_{\text{L1 regularization}} = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2 + \alpha \|\boldsymbol{\beta}\|_1$$

Note:

- $\|\boldsymbol{\beta}\| = \|\boldsymbol{\beta}\|_2$ is known as the "L2 norm" or Euclidean norm
- $\|\boldsymbol{\beta}\|_1$ is known as the "L1 norm" or "taxi-cab" norm or "Manhattan" norm

## Ridge regression

- Recall the ridge cost:

$$J_{\text{ridge}}(\boldsymbol{\beta}) = \text{RSS}(\boldsymbol{\beta}) + \alpha \sum_{j=1}^{d} |\beta_j|^2 = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2 + \alpha\|\boldsymbol{\beta}\|^2$$

- Similar to the derivation of $\boldsymbol{\beta}_{\text{ls}}$, can show that

$$\boldsymbol{\beta}_{\text{ridge}} \triangleq \arg\min_{\boldsymbol{\beta}} J_{\text{ridge}}(\boldsymbol{\beta}) = (\boldsymbol{X}^\top\boldsymbol{X} + \alpha\boldsymbol{I})^{-1}\boldsymbol{X}^\top\boldsymbol{y}$$

- Why use the penalty $\|\boldsymbol{\beta}\|^2$?
  - When columns of $\boldsymbol{X}$ are correlated, the unregularized LS solution $\boldsymbol{\beta}_{\text{ls}} = (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y}$ can have very large values due to the inverse
  - Equivalently, feature correlation can make the cost surface $\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2$ very stretched and its minimum, $\boldsymbol{\beta}_{\text{ls}}$, very far from the origin
  - Problem: large $\boldsymbol{\beta}_{\text{ls}}$ implies $\widehat{y} = \boldsymbol{\beta}_{\text{ls}}^\top\boldsymbol{x}$ is sensitive to test data $\boldsymbol{x}$ (i.e., overfitting)
  - By penalizing $\|\boldsymbol{\beta}\|^2$, we discourage large $\boldsymbol{\beta}$ and thus help to reduce overfitting

## Coefficient path of ridge regression

- The "coefficient path" is the plot of all $\beta_j$ versus the regularization strength $\alpha$
- With ridge regression, larger $\alpha$ leads to smaller $\beta_j$ but *not fewer* non-zero $\beta_j$
- Choose $\alpha$ via cross-validation

Figure from book: Hastie, Tibshirani, Friedman, *The Elements of Statistical Learning*

$\leftarrow$ larger $\alpha$ smaller $\alpha \rightarrow$

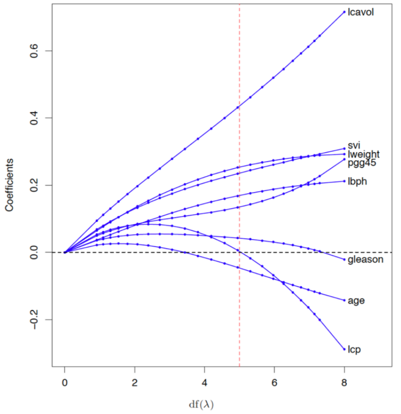


**FIGURE 3.8.** *Profiles of ridge coefficients for the prostate cancer example, as the tuning parameter $\lambda$ is varied. Coefficients are plotted versus $\mathrm{df}(\lambda)$, the effective degrees of freedom. A vertical line is drawn at $\mathrm{df} = 5.0$, the value chosen by cross-validation.*

## LASSO

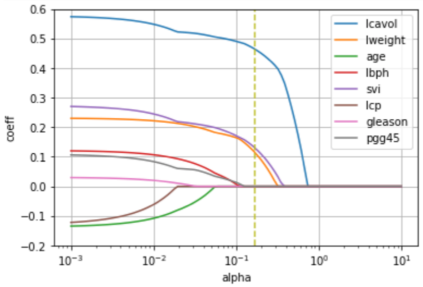
- Recall LASSO cost:

$$J_{\text{lasso}}(\boldsymbol{\beta}) = \text{RSS}(\boldsymbol{\beta}) + \alpha \textstyle\sum_{j=1}^{d} |\beta_j|_1 = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2 + \alpha\|\boldsymbol{\beta}\|_1$$

- No closed-form expression for $\boldsymbol{\beta}_{\text{lasso}} \triangleq \arg\min_{\boldsymbol{\beta}} J_{\text{lasso}}(\boldsymbol{\beta})$
  - but convex optimization problem $\Rightarrow$ tractable solution
  - many fast numerical solvers: FISTA, ADMM, glmnet
  - implemented in sklearn via the Lasso method
- Why use penalty $\|\boldsymbol{\beta}\|_1$?
  - Leads to exactly zero $\beta_j$, and thus feature selection!
    - $\alpha$ controls # of nonzero $\beta_j$
  - Careful: the non-zero $\beta_j$ are biased towards 0
    - discard them and keep only the *indices* of informative features, $\{j : \beta_j \neq 0\}$
    - then do standard linear regression (i.e., LS) with the informative $x_j$

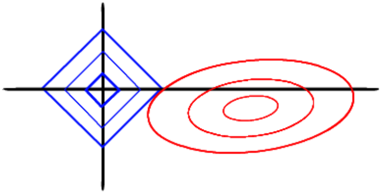
## Coefficient path of LASSO

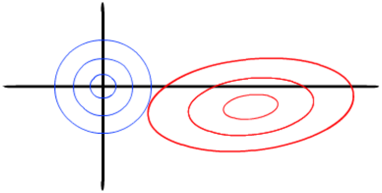
- The "coefficient path" is the plot of all $\beta_j$ versus the regularization strength $\alpha$
- With LASSO, larger $\alpha$ leads to fewer non-zero $\beta_j$
- The LASSO path suggests which features $\{x_j\}$ are most informative
  - In the PSA demo, LASSO suggests `lcavol` is most informative
- Choose $\alpha$ via cross-validation



## Summary of ridge regression and LASSO

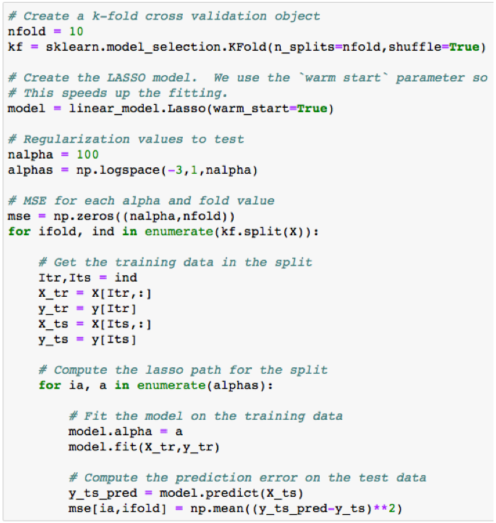
- LASSO (L1 penalty)
  - Tends to produce many exactly-zero $\beta_j$
  - Great for feature selection!
  - But no closed-form solution: solve numerically
- Ridge regression (L2 penalty)
  - Can write solution in closed-form
  - Tends to produce many small $\beta_j$
  - Not useful for feature selection
  - But helps with correlated features



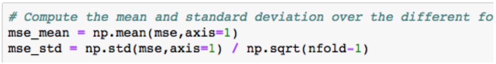


## Implementing LASSO with sklearn

- sklearn has a Lasso method
- On right, we choose the regularization weight $\alpha$ using K-fold cross-validation:
  - Outer loop over folds $k$
  - Inner loop over a grid of $\alpha$
  - First compute $\text{RSS}_{k,\alpha}$ for all $k$ and $\alpha$
  - Then compute $\overline{\text{RSS}}_{\alpha}$ and $\text{SE}_{\alpha}$ for each $\alpha$



```
# Create a k-fold cross validation object
nfold = 10
kf = sklearn.model_selection.KFold(n_splits=nfold,shuffle=True)

# Create the LASSO model.  We use the `warm start` parameter so
# This speeds up the fitting.
model = linear_model.Lasso(warm_start=True)

# Regularization values to test
nalpha = 100
alphas = np.logspace(-3,1,nalpha)

# MSE for each alpha and fold value
mse = np.zeros((nalpha,nfold))
for ifold, ind in enumerate(kf.split(X)):

    # Get the training data in the split
    Itr,Its = ind
    X_tr = X[Itr,:]
    y_tr = y[Itr]
    X_ts = X[Its,:]
    y_ts = y[Its]

    # Compute the lasso path for the split
    for ia, a in enumerate(alphas):

        # Fit the model on the training data
        model.alpha = a
        model.fit(X_tr,y_tr)

        # Compute the prediction error on the test data
        y_ts_pred = model.predict(X_ts)
        mse[ia,ifold] = np.mean((y_ts_pred-y_ts)**2)
```

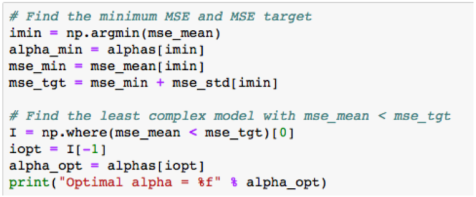


```
# Compute the mean and standard deviation over the different fo
mse_mean = np.mean(mse,axis=1)
mse_std = np.std(mse,axis=1) / np.sqrt(nfold-1)
```

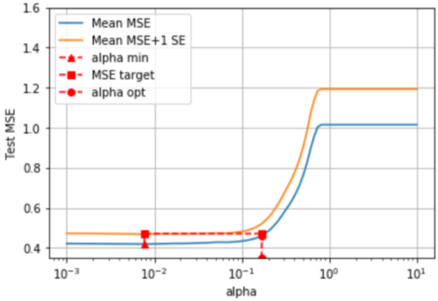
## Applying the one-standard-error rule

Select $\alpha$ via one-standard-error rule:

- Find $\alpha_{\min} = \arg\min_\alpha \overline{\text{RSS}}_\alpha$
- Set $\overline{\text{RSS}}_{\textbf{tgt}} = \overline{\text{RSS}}_{\alpha_{\min}} + \text{SE}_{\alpha_{\min}}$
- Find simplest model (largest $\alpha$) such that $\overline{\text{RSS}}_\alpha < \overline{\text{RSS}}_{\textbf{tgt}}$

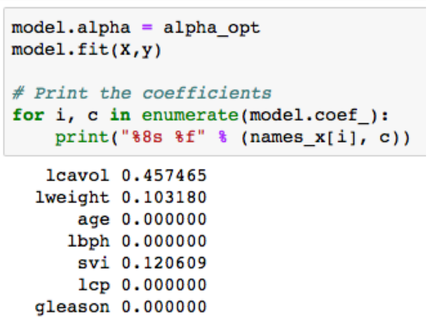


```
# Find the minimum MSE and MSE target
imin = np.argmin(mse_mean)
alpha_min = alphas[imin]
mse_min = mse_mean[imin]
mse_tgt = mse_min + mse_std[imin]

# Find the least complex model with mse_mean < mse_tgt
I = np.where(mse_mean < mse_tgt)[0]
iopt = I[-1]
alpha_opt = alphas[iopt]
print("Optimal alpha = %f" % alpha_opt)
```



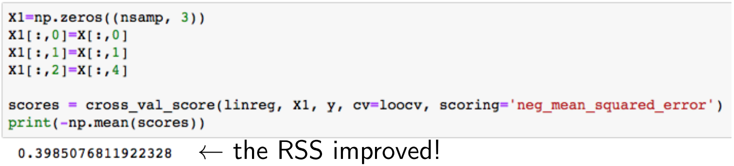
## Using LASSO for feature selection

- Having found $\alpha$, we next compute the LASSO coefficients $\beta_{\text{lasso}}$ on the full training data
  - LASSO selects only 3 features for prediction!
- We then isolate these features and use them for LS-based linear regression
  - Note $\beta_{\text{lasso}}$ is used only for feature selection
  - Why? Non-zeros in $\beta_{\text{lasso}}$ are biased



```
model.alpha = alpha_opt
model.fit(X,y)

# Print the coefficients
for i, c in enumerate(model.coef_):
    print("%8s %f" % (names_x[i], c))
```

```
  lcavol 0.457465
 lweight 0.103180
     age 0.000000
    lbph 0.000000
     svi 0.120609
     lcp 0.000000
 gleason 0.000000
```



```
X1=np.zeros((nsamp, 3))
X1[:,0]=X[:,0]
X1[:,1]=X[:,1]
X1[:,2]=X[:,4]

scores = cross_val_score(linreg, X1, y, cv=loocv, scoring='neg_mean_squared_error')
print(-np.mean(scores))
```

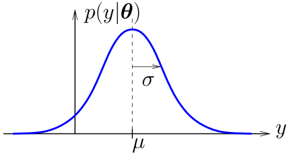
```
0.3985076811922328
```

$\leftarrow$ the RSS improved!

## Outline

## Estimation of statistical parameters

- At its core, machine *learning* is essentially about estimating statistical parameters in a probabilistic model of the data
- We now describe the maximum-likelihood (ML) and maximum a posteriori (MAP) approaches to this important problem
- Say $y$ is a random variable that depends on some statistical parameters $\boldsymbol{\theta}$.
  - Then $y$ is fully described by its probability density function (pdf) $p(y|\boldsymbol{\theta})$.
  - To visualize the pdf, we usually plot $p(y|\boldsymbol{\theta})$ versus $y$ for some hypothesized $\boldsymbol{\theta}$
- Example: $y \sim \mathcal{N}(\mu, \sigma^2)$, i.e., $y$ is Gaussian with mean $\mu$ and variance $\sigma^2$
  - Then $p(y|\boldsymbol{\theta}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{1}{2\sigma^2}(y-\mu)^2)$ with parameters $\boldsymbol{\theta} = [\mu, \sigma^2]^\mathsf{T}$
  - This pdf is a "bell curve" centered at $\mu$ with width $\sigma$:



## Maximum-likelihood (ML) estimation

- Now say that we observe independent samples $\boldsymbol{y} = [y_1, \dots, y_n]^\mathsf{T}$ of a random variable $y$ with pdf $p(y|\boldsymbol{\theta})$.

- For this fixed $\boldsymbol{y}$, the function $p(\boldsymbol{y}|\boldsymbol{\theta})$ versus $\boldsymbol{\theta}$ is called the "likelihood function"

- Since we've assumed $\{y_i\}_{i=1}^n$ are independent & identically distributed (i.i.d.),
$$p(\boldsymbol{y}|\boldsymbol{\theta}) = \prod_{i=1}^n p(y_i|\boldsymbol{\theta})$$

- A common way of estimating $\boldsymbol{\theta}$ from observed $\boldsymbol{y}$ is to maximize the likelihood:
$$\boldsymbol{\theta}_{\mathsf{ml}} \triangleq \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{y}|\boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \{\ln p(\boldsymbol{y}|\boldsymbol{\theta})\} = \arg\min_{\boldsymbol{\theta}} \{-\ln p(\boldsymbol{y}|\boldsymbol{\theta})\}$$
where $\ln(\cdot)$ and negation are often used to simplify the expression

- ML estimation uses no prior belief about $\boldsymbol{\theta}$; it only fits the observed data

## ML estimation for linear regression

- Can we use ML estimation to fit the parameters $\beta$ of linear regression? Yes!
- Suppose that our data obeys the linear-Gaussian model
$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} \;\text{ with }\; \{\epsilon_i\} \sim \text{i.i.d. } \mathcal{N}(0, \sigma^2)$$
assuming standardized data, and thus $\boldsymbol{\beta} = [\beta_1, \dots, \beta_d]^\mathsf{T}$ (i.e., no intercept $\beta_0$)
- For this model, can show that the likelihood fxn becomes
$$p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\beta}) = \frac{\exp(-\frac{1}{2\sigma^2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2)}{(2\pi\sigma^2)^{n/2}}$$
- The ML estimate is then
$$\boldsymbol{\beta}_{\mathsf{ml}} = \arg\min_{\boldsymbol{\beta}} \big\{ -\ln p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\beta}) \big\} = \arg\min_{\boldsymbol{\beta}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2 = \boldsymbol{\beta}_{\mathsf{ls}}$$
- So, under this linear-Gaussian model, the ML estimate is the least-squares fit!

## Maximum a posteriori (MAP) estimation

- To incorporate a prior belief on $\boldsymbol{\beta}$, we can use MAP estimation
- The MAP estimate of $\boldsymbol{\beta}$ from $\boldsymbol{y}$ is
$$\boldsymbol{\beta}_{\mathsf{map}} \triangleq \arg\max_{\boldsymbol{\beta}} p(\boldsymbol{\beta}|\boldsymbol{X},\boldsymbol{y})$$
i.e., the most probable $\boldsymbol{\beta}$ given the data $(\boldsymbol{X},\boldsymbol{y})$
- Bayes rule says
$$p(\boldsymbol{\beta}|\boldsymbol{X},\boldsymbol{y}) = \frac{p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\beta})p(\boldsymbol{\beta})}{p(\boldsymbol{y})} \text{ for ``prior'' pdf } p(\boldsymbol{\beta})$$
which implies that
$$\boldsymbol{\beta}_{\mathsf{map}} = \arg\max_{\boldsymbol{\beta}} \frac{p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\beta})p(\boldsymbol{\beta})}{p(\boldsymbol{y})} = \arg\max_{\boldsymbol{\beta}} \left\{p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\beta})p(\boldsymbol{\beta})\right\}$$
- Interpretations:

| | |
|---|---|
| likelihood $p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\beta})$: | how well $\boldsymbol{\beta}$ agrees with data $(\boldsymbol{X},\boldsymbol{y})$ |
| prior $p(\boldsymbol{\beta})$: | how well $\boldsymbol{\beta}$ agrees with prior belief |

- Key point: MAP estimation uses both likelihood *and* prior

## Regularized linear regression is MAP estimation

- It's often simpler to formulate
$$\begin{aligned}\boldsymbol{\beta}_{\text{map}} &= \arg\max_{\boldsymbol{\beta}} \left\{ p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\beta})p(\boldsymbol{\beta}) \right\} \\ &= \arg\min_{\boldsymbol{\beta}} \left\{ -\ln\left[ p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\beta})p(\boldsymbol{\beta})\right] \right\} \\ &= \arg\min_{\boldsymbol{\beta}} \left\{ -\ln p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\beta}) - \ln p(\boldsymbol{\beta}) \right\}\end{aligned}$$

- The linear-Gaussian model: $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$ with i.i.d. $\epsilon_i \sim \mathcal{N}(0,\sigma^2)$, gives
$$-\ln p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\beta}) = \tfrac{1}{2\sigma^2}\|\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}\|^2 + \text{const}$$
and so
$$\begin{aligned}\boldsymbol{\beta}_{\text{map}} &= \arg\min_{\boldsymbol{\beta}} \left\{ \tfrac{1}{2\sigma^2}\|\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}\|^2 - \ln p(\boldsymbol{\beta}) \right\} \\ &= \arg\min_{\boldsymbol{\beta}} \Big\{ \underbrace{\|\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}\|^2}_{\mathrm{RSS}(\boldsymbol{\beta})} \underbrace{-2\sigma^2 \ln p(\boldsymbol{\beta})}_{+\ \phi(\boldsymbol{\beta})} \Big\}\end{aligned}$$

- Thus MAP estimation under the linear-Gaussian model is equivalent to regularized linear regression!

## MAP interpretations of Ridge Regression and LASSO

- Recall that

$$\boldsymbol{\beta}_{\text{map}} = \arg\min_{\boldsymbol{\beta}} \left\{ \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2 - 2\sigma^2 \ln p(\boldsymbol{\beta}) \right\}$$

- If our prior belief is that $\beta_j$ is i.i.d. $\mathcal{N}(0, v)$, then

$$p(\boldsymbol{\beta}) = \prod_{j=1}^{d} \frac{\exp(-\frac{1}{2v}(\beta_j)^2)}{\sqrt{2\pi v}} \quad \Rightarrow \quad \ln p(\boldsymbol{\beta}) = -\tfrac{1}{2v}\|\boldsymbol{\beta}\|^2 + \text{const}$$

$$\Rightarrow \boldsymbol{\beta}_{\text{map}} = \arg\min_{\boldsymbol{\beta}} \left\{ \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2 + \tfrac{\sigma^2}{v}\|\boldsymbol{\beta}\|^2 \right\} \;\Leftrightarrow\; \text{Ridge Regression, } \alpha = \tfrac{\sigma^2}{v}$$

- If our prior belief is that $\beta_j$ is i.i.d. Laplacian$(0, \lambda)$, then

$$p(\boldsymbol{\beta}) = \prod_{j=1}^{d} \frac{\exp(-\frac{1}{\lambda}|\beta_j|)}{2\lambda} \quad \Rightarrow \quad \ln p(\boldsymbol{\beta}) = -\tfrac{1}{\lambda}\|\boldsymbol{\beta}\|_1 + \text{const}$$

$$\Rightarrow \boldsymbol{\beta}_{\text{map}} = \arg\min_{\boldsymbol{\beta}} \left\{ \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2 + \tfrac{2\sigma^2}{\lambda}\|\boldsymbol{\beta}\|_1 \right\} \;\Leftrightarrow\; \text{LASSO, } \alpha = \tfrac{2\sigma^2}{\lambda}$$

## Outline

## Linear regression with vector-valued targets

- Until now we've focused on linearly predicting a scalar-valued target $y_i$ from features $\boldsymbol{x}_i \in \mathbb{R}^d$ using a coefficient vector $\boldsymbol{\beta} \in \mathbb{R}^{d+1}$ with intercept term $\beta_0$:

$$y_i \approx \begin{bmatrix} 1 & \boldsymbol{x}_i^\mathsf{T} \end{bmatrix} \boldsymbol{\beta}$$

- We can extend this approach to vector-valued targets $\boldsymbol{y}_i^\mathsf{T} \in \mathbb{R}^K$ as follows

$$\boldsymbol{y}_i^\mathsf{T} \triangleq \begin{bmatrix} y_{i1} & \cdots & y_{iK} \end{bmatrix} \approx \begin{bmatrix} 1 & \boldsymbol{x}_i^\mathsf{T} \end{bmatrix} \underbrace{\begin{bmatrix} \boldsymbol{\beta}_1 & \cdots & \boldsymbol{\beta}_K \end{bmatrix}}_{\triangleq \boldsymbol{B}}$$

where now we use a coefficient matrix $\boldsymbol{B} \in \mathbb{R}^{(d+1)\times K}$.

- Incorporating all of the training samples $i = 1, \dots, n$, we get the model

$$\underbrace{\begin{bmatrix} \boldsymbol{y}_1^\mathsf{T} \\ \vdots \\ \boldsymbol{y}_n^\mathsf{T} \end{bmatrix} = \begin{bmatrix} y_{11} & \cdots & y_{1K} \\ \vdots & & \vdots \\ y_{n1} & \cdots & y_{nK} \end{bmatrix}}_{\triangleq \boldsymbol{Y}} \approx \underbrace{\begin{bmatrix} 1 & \boldsymbol{x}_1^\mathsf{T} \\ \vdots & \vdots \\ 1 & \boldsymbol{x}_n^\mathsf{T} \end{bmatrix}}_{\boldsymbol{A}} \underbrace{\begin{bmatrix} \boldsymbol{\beta}_1 & \cdots & \boldsymbol{\beta}_K \end{bmatrix}}_{\boldsymbol{B}}$$

## Regularized regression with vector-valued targets

- With a linear-Gaussian model and standardized data (no intercepts), we get

$$\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{B} + \boldsymbol{E} \quad \text{with i.i.d. } \epsilon_{ik} \sim \mathcal{N}(0, \sigma^2)$$

- For Ridge Regression, we would solve for

$$\boldsymbol{B}_{\text{ridge}} \triangleq \arg\min_{\boldsymbol{B}} \left\{ \|\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{B}\|_F^2 + \alpha \|\boldsymbol{B}\|_F^2 \right\}$$

  where $\|\boldsymbol{B}\|_F^2 \triangleq \sum_{j,k} b_{jk}^2$ is the (squared) matrix Frobenius norm

  - We don't use the matrix norm $\|\boldsymbol{B}\| = \|\boldsymbol{B}\|_2$, which has a different meaning!

- Meanwhile, for LASSO, we would solve for

$$\boldsymbol{B}_{\text{lasso}} \triangleq \arg\min_{\boldsymbol{B}} \left\{ \|\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{B}\|_F^2 + \alpha \|\boldsymbol{B}\|_1 \right\}$$

  where $\|\boldsymbol{B}\|_1 \triangleq \sum_{j,k} |b_{jk}|$ is the matrix L1 norm

- sklearn's linear regression methods have no trouble with these extensions

## Learning objectives

- Understand motivation for and concept of feature selection
- Understand feature selection methods based on:
  - exhaustive search
  - stepwise selection
  - target cross-correlation
  - regularization
- Understand ridge regression and LASSO:
  - interpret their coefficient paths
  - implement LASSO using sklearn
  - know how to select the regularization strength using cross-validation
- Understand connections to ML estimation and MAP estimation
- Understand how to handle vector-valued targets