Unit 4
Feature Selection, LASSO, and Maximum Likelihood

Prof. Phil Schniter

THE OHIO STATE UNIVERSITY

ECE 5307: Introduction to Machine Learning, Sp23

Learning objectives

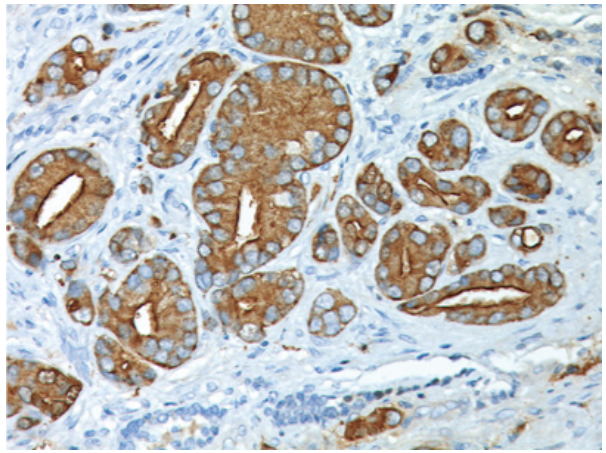
- Understand motivation and idea behind feature selection
- Understand feature selection methods based on:
  - exhaustive search
  - stepwise selection
  - univariate statistics
  - regularization
- Understand ridge regression and LASSO:
  - interpret their coefficient paths
  - implement LASSO using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

Outline

- Motivating Example: Predicting Prostate Cancer
- Feature Selection
- Ridge Regression and LASSO
- Maximum Likelihood and MAP
- Regression with Vector-Valued Targets

Prostate-specific antigen (PSA) testing

- High PSA is an indicator of prostate cancer
  - PSA is a common tool for screening
- Famous 1989 study by Starney et al.:
  - Measured PSA level of 97 men prior to prostate removal
  - Measured 8 biometrics, including cancer volume, prostate weight, age, etc.
- Machine-learning problem:
  - Can we predict PSA from these biometrics?
  - Which biometrics are most important?



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 is 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: LS Linear Regression

- Let's try first with multiple linear regression:

$$y \approx \widehat{y} = \widehat{\beta}_0 + \widehat{\beta}_1 x_1 + \cdots + \widehat{\beta}_d x_d$$

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

- Why linear regression?
  - Coefficients are easy to fit (via LS)
  - Coefficients are easy to interpret
    - larger $|\widehat{\beta}_j|$ means x_j has larger effect on PSA (if x_j are roughly same size)

```
from sklearn import linear_model

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

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(linreg, Xtr, ytr, cv=rkf, scoring='neg_mean_squared_error')
mse_cv_ls = -np.mean(scores)
```

```
cross-validation MSE = 0.404849
```

- Can we do better?

Outline

- Motivating Example: Predicting Prostate Cancer
- Feature Selection
- Ridge Regression and LASSO
- Maximum Likelihood and MAP
- Regression with Vector-Valued Targets

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)

- Option 1: Forward selection
  - First use cross-validation to find the single feature yielding the lowest test RSS
  - Then, add one of the remaining features so that the pair provides the lowest RSS
  - Repeat until RSS starts to increase, or some max allowed # features is reached
- Option 2: Backwards elimination
  - First use all d features and compute the test RSS using CV
  - Then, remove one feature, so that the remaining features give the lowest RSS
  - Repeat until RSS starts to increase, or some min allowed # of features is reached
- Both approaches are suboptimal
  - They are "greedy" or "myopic": only look one step ahead
  - Still, they often work well
  - Backwards elimination is implemented in sklearn.RFECV

Feature selection via univariate statistics

Here is another suboptimal strategy. Again, assume d total features.

- Maximize correlation with target:
  - For each $p \in \{1, \ldots, d\}$, find the p features that are most correlated with the target (i.e., compute s_{yx_j} for each j, and choose the p values of j giving largest $|s_{yx_j}|$)
  - 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
- Alternative: Maximize some other univariate statistic that quantifies how related a feature is to the target
  - Examples: mutual information, F-value, P-value, etc.
  - Many are implemented in sklearn.SelectKBest

Feature selection via regularization

- A third (suboptimal) strategy is regularization: add a term to the cost function that penalizes the use of many features
- For example, if the unregularized training objective is

$$\widehat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \mathrm{RSS}(\boldsymbol{\beta})$$

  then the regularized training objective is

$$\widehat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \left\{ \mathrm{RSS}(\boldsymbol{\beta}) + \phi(\boldsymbol{\beta}) \right\}$$

  where $\phi(\boldsymbol{\beta})$ is the regularization or penalty term
- In other words, to justify using the jth feature, the training $\mathrm{RSS}(\boldsymbol{\beta})$ must decrease more than the penalty $\phi(\boldsymbol{\beta})$ increases
- We'll focus on this approach for the remainder of the unit

Outline

- Motivating Example: Predicting Prostate Cancer
- Feature Selection
- Ridge Regression and LASSO
- Maximum Likelihood and MAP
- Regression with Vector-Valued Targets

Regularized RSS

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

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

- Can we modify this to perform feature selection?
- Idea: Penalize the use of each feature (i.e., penalize $\beta_j \neq 0$ for all j)
  - Feature j should not be used unless it significantly reduces RSS
  - In other words, set $\widehat{\beta}_j = 0$ unless RSS is significantly reduced by $\widehat{\beta}_j \neq 0$
  - To do this, add a regularization term $\phi(\boldsymbol{\beta})$ to the optimization objective:

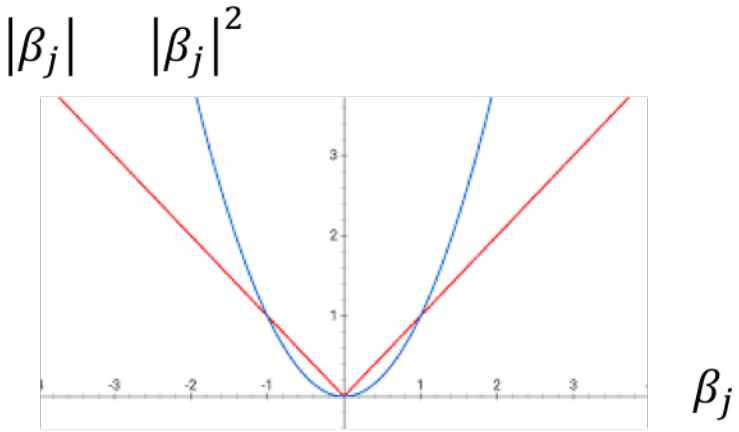
$$\widehat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} J(\boldsymbol{\beta}) \quad \text{for} \quad J(\boldsymbol{\beta}) = \text{RSS}(\boldsymbol{\beta}) + \phi(\boldsymbol{\beta})$$

    where $\phi(\cdot)$ encourages $\widehat{\beta}_j = 0$ for many j
  - How should we choose the function $\phi(\cdot)$?

L1 and L2 regularization

- L2 regularization: $\phi(\boldsymbol{\beta}) = \alpha \sum_{j=1}^{d} |\beta_j|^2$

- L1 regularization: $\phi(\boldsymbol{\beta}) = \alpha \sum_{j=1}^{d} |\beta_j|$



- Both penalize $\beta_j \neq 0$, but they act in different ways
- The overall strength of the penalty is controlled by $\alpha \geq 0$
- Note: regularization is *not* applied to the intercept term β_0. It does not multiply a feature x_j, so we do *not* want to penalize it!

Standardize your data!

- Motivation:
  - The L1 and L2 regularizers penalize all coefficients β_j *uniformly*
  - But if some x_j are relatively large, then the corresponding β_j may be relatively small, in which case uniform penalization may not work well
  - Can avoid this issue by normalizing the x_j (i.e., set $\frac{1}{n}\sum_i x_{ij}^2 = \overline{x}_j^2 + s_{x_j}^2 = 1 \; \forall j$)
  - If we also remove the mean of y and each x_j, then conveniently we get $\widehat{\beta}_0 = 0$
- 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 training data

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
Xtr = scaler.fit_transform(X0)
ytr = scaler.fit_transform(y0.reshape(-1,1)).reshape(-1) # vector -> matrix and
```

  - Standardize test data using the training statistics, not test statistics!

Ridge regression and LASSO

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

- Ridge regression cost function:

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

- LASSO cost function:

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

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

Ridge regression

- Recall the ridge cost:

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

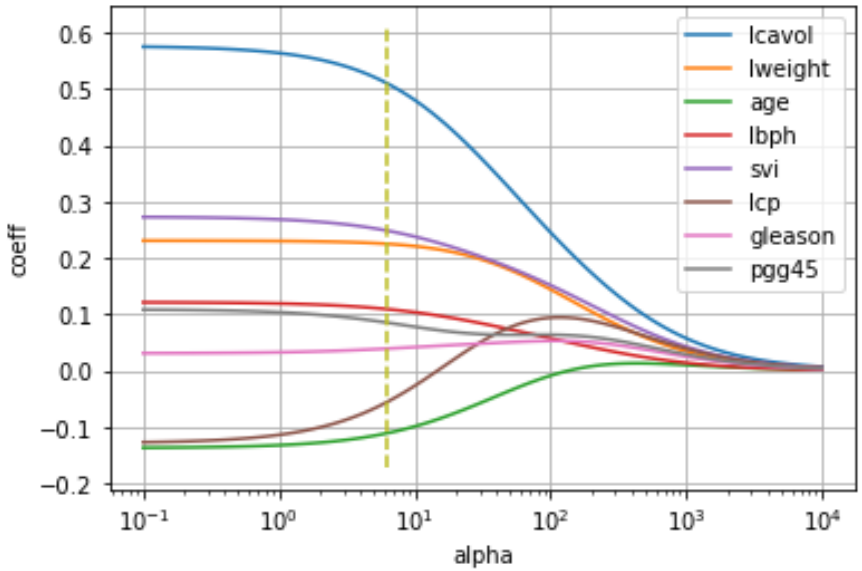
- Similar to how we derived $\boldsymbol{\beta}_{\text{ls}}$, one can show that

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

- Why use the L2 penalty $\|\boldsymbol{\beta}\|^2$?
  - When columns of $\boldsymbol{X}$ are correlated, the unregularized LS solution $\boldsymbol{\beta}_{\text{ls}} = (\boldsymbol{X}^{\mathsf{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathsf{T}}\boldsymbol{y}$ 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
  - Large $\boldsymbol{\beta}_{\text{ls}}$ means $\widehat{y} = \boldsymbol{\beta}_{\text{ls}}^{\mathsf{T}}\boldsymbol{x}$ will be 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 β_j versus the regularization strength α
- With ridge regression, larger α leads to smaller β_j but *not fewer* non-zero β_j
- Choose α via cross-validation



LASSO

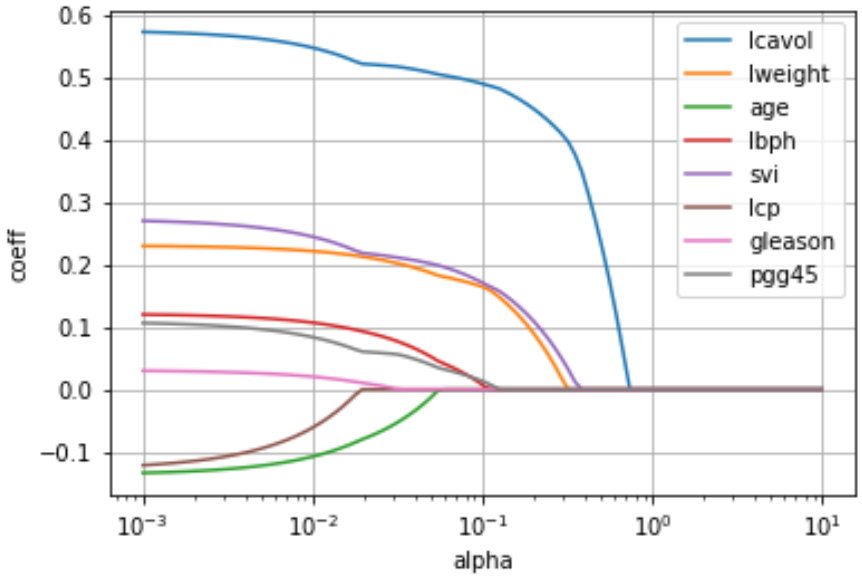
- Recall LASSO cost:

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

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

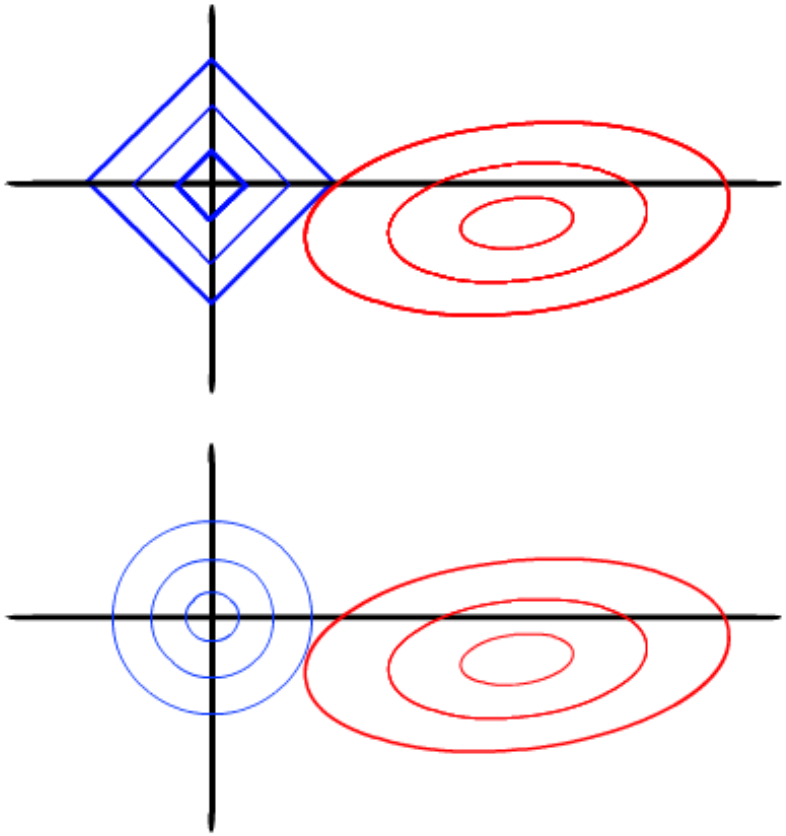
Coefficient path of LASSO

- The "coefficient path" is the plot of all β_j versus the regularization strength α
- With LASSO, larger α leads to...
  1. fewer non-zero β_j
  2. smaller amplitudes $|\beta_j|$
- The LASSO path suggests which features $\{x_j\}$ are most informative
  - In the PSA demo, LASSO suggests `lcavol` is most informative
- Choose α via cross-validation
  - We'll try several variations



Summary of ridge regression and LASSO

- LASSO (L1 penalty)
  - Tends to produce many exactly-zero β_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 β_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 α using K-fold cross-validation:
  - Outer loop over folds k
  - Inner loop over a grid of α
  - First compute $\mathrm{RSS}_{k,\alpha}$ for all k & α
  - Then compute $\overline{\mathrm{RSS}}_\alpha$ and SE_α for each α
- Demo also shows how to implement this in a much simpler manner using sklearn.GridSearchCV

```
# Shortcut using GridSearchCV
from sklearn.model_selection import GridSearchCV

# Select which estimator parameters to optimize
param_grid = {'alpha': alphas}

# Run cross-validation
gscv = GridSearchCV(lasso, param_grid, cv=kf, scoring='neg_mean_squared_error')
gscv.fit(Xtr,ytr)
mse_cv = -gscv.cv_results_['mean_test_score']
mse_se = gscv.cv_results_['std_test_score']/np.sqrt(nfold-1) # note division by (nfold-1)!
```

```
# Manual approach using 2 for-loops

# 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 s
# This speeds up the fitting.
lasso = 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(Xdes)):

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

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

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

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

# Compute the MSE mean over the folds and its standard error
mse_cv2 = np.mean(mse,axis=1)
mse_se2 = np.std(mse,axis=1,ddof=1) / np.sqrt(nfold)
```

Using LASSO for feature selection with LS (naive method)

- Naive approach: Choose α to minimize $\overline{\mathrm{RSS}}_\alpha$ (or use the OSE rule)
- Then compute the LASSO coefficients $\boldsymbol{\beta}_{\text{lasso}}$ on the full training data
  - LASSO selects 7 of the 8 features for prediction
- Then isolate these features and use them for LS-based linear regression
  - Note $\boldsymbol{\beta}_{\text{lasso}}$ is used only for feature selection
  - Why? Non-zeros in $\boldsymbol{\beta}_{\text{lasso}}$ are biased

```
# Refit LASSO using CV-tuned alpha
lasso.alpha = alpha_min
lasso.fit(Xtr,ytr)

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

# Print the cross-validation MSE
print("\ncross-validation MSE = %f" % mse_cv_lasso)
```

```
LASSO coefficients:
 intrcpt 0.000000
  lcavol 0.519706
 lweight 0.207187
     age -0.065792
    lbph 0.084529
     svi 0.214778
     lcp -0.000000
 gleason 0.005118
   pgg45 0.058145

cross-validation MSE = 0.398888 ← poor!
```

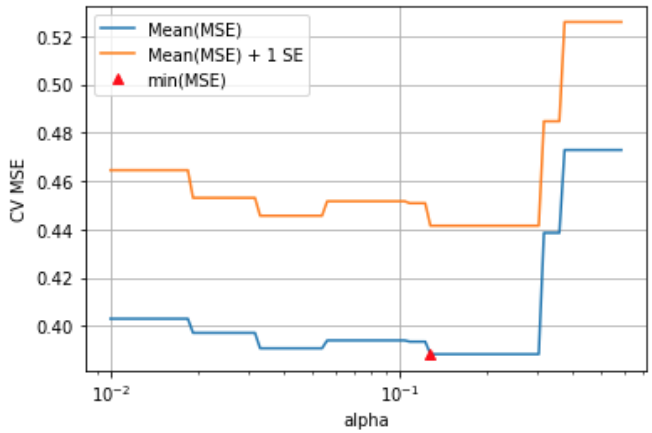
```
# Train LASSO-feature-selected LS on entire design data
subset = np.where(np.abs(lasso.coef_)>0)[0]

# Print the cross-validation MSE
scores = cross_val_score(linreg, Xtr[:, subset], ytr, cv=rkf, scoring='neg_mean_squared_erro
mse_cv_lasso_ls = -np.mean(scores)
```

```
cross-validation MSE = 0.397190 ← improved!
```

Tuning LASSO+LS using a grid search (via for-loop)

- We can do better!
- The α on the previous page was selected to minimize the RSS of $\boldsymbol{\beta}_{\text{lasso}}$, but we don't actually use $\boldsymbol{\beta}_{\text{lasso}}$ for prediction! (Only feature selection)
- We should instead tune α to minimize the RSS of the *LASSO-selected LS coefficients*! On the right, we do this with a for-loop over α
- As α increases, LASSO sets more coefficients to zero, thereby selecting fewer features. We use cross-validation to estimate the performance of a LS fit of those selected features

```
# compute the CV MSE of LASSO+LS over a grid of alpha
alphas = np.logspace(-2,-0.23,100)
mse_cv = np.zeros(len(alphas))
mse_se = np.zeros(len(alphas))
for i,a in enumerate(alphas):
    # fit LASSO
    lasso.set_params(alpha = a)
    lasso.fit(Xtr,ytr)
    # select features
    nonzero = (np.abs(lasso.coef_)>1e-5) # logical array
    subset = np.where(nonzero)[0] # array of feature indices
    # fit LS and evaluate performance using CV
    mse = -cross_val_score(linreg, Xtr[:, subset], ytr, cv=rkf,
                                        scoring='neg_mean_squared_error')
    mse_cv[i] = np.mean(mse)
    mse_se[i] = np.std(mse,ddof=1) / np.sqrt(nfold)
```

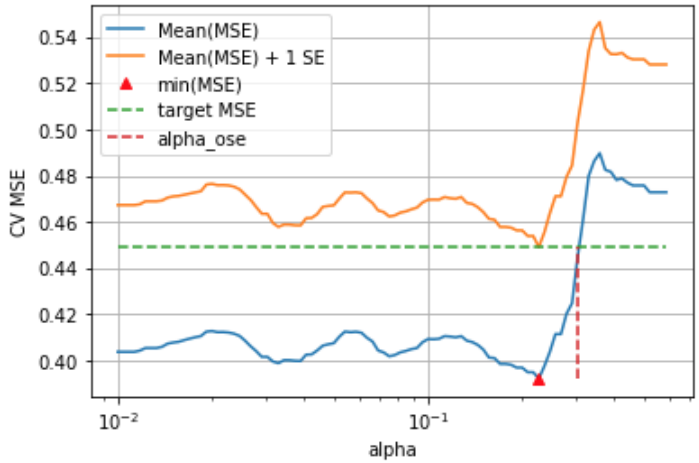


`cross-validation MSE = 0.388373` ← improved!

Tuning LASSO+LS using a grid search (via pipeline)

- Instead of a for-loop, we can use sklearn.GridSearchCV
  - Need to specify estimator, parameter grid, cross-val object, & performance metric
  - But here the "estimator" is the cascade: LASSO $\rightarrow$ feature selection $\rightarrow$ LS
  - Can implement this cascade using an sklearn pipeline

```
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectFromModel

# build pipeline for LASSO+FeatureSelect+LS
pipe = Pipeline([
  ('LASSO',SelectFromModel(linear_model.Lasso(fit_intercept=False,warm_start=False))),
  ('LS', linear_model.LinearRegression(fit_intercept=False))
])

# tune LASSO regularization strength
parameters = {'LASSO__estimator__alpha':alphas} # see pipe.get_params().keys()
gscv = GridSearchCV(pipe, parameters, cv=kf, scoring='neg_mean_squared_error')
gscv.fit(Xtr,ytr);
mse_cv = -gscv.cv_results_['mean_test_score']
mse_se = gscv.cv_results_['std_test_score']/np.sqrt(nfold-1) # note division by (nfold-1)

# Find the minimum MSE
imin = np.argmin(mse_cv)
alpha_min = alphas[imin]
mse_cv_lasso_ls2 = mse_cv[imin]
```




```
cross-validation MSE = 0.392121
      with OSE rule = 0.388373
```

- Note: Results are slightly different than with for-loop. For each α...
  - for-loop method runs LASSO once on full data, then evaluates LS using K-fold CV
  - pipeline evaluates LASSO+LS using K-fold CV

Summary of CV results from demo

- Results are as expected:
  - exhaustive-search LS is best
  - plain LS (no feature selection) is much worse
  - LASSO & Ridge plagued by coefficient bias
  - debiased LASSO slightly better
  - LASSO+LS very good (nearly optimal!)
  - RFE-CV+LS also very good
  - RFE-MI+LS pretty bad

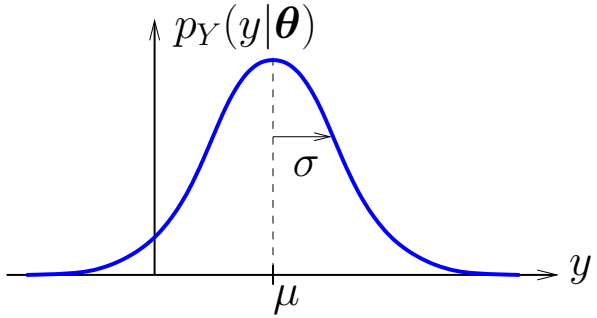
```
cross-validation MSE:
  LS = 0.403048
  Ridge = 0.398266
  LASSO = 0.398888
  LASSO(debiased) = 0.397190
  LASSO+LS(for)= 0.388373
  LASSO+LS(pipe)= 0.388373
  RFE-CV+LS = 0.388373
  RFE-MI+LS = 0.403048
  exhaustive+LS = 0.388072
```

Outline

- Motivating Example: Predicting Prostate Cancer
- Feature Selection
- Ridge Regression and LASSO
- Maximum Likelihood and MAP
- Regression with Vector-Valued Targets

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 unknown statistical parameters $\boldsymbol{\theta}$
  - Then Y is fully described by its probability density function (pdf) $p_Y(\cdot|\boldsymbol{\theta})$
  - To visualize the pdf, we usually plot $p_Y(y|\boldsymbol{\theta})$ versus y for some hypothesized $\boldsymbol{\theta}$
- Example: Gaussian $Y \sim \mathcal{N}(\mu, \sigma^2)$ has unknown mean μ and variance σ^2
  - Then $p_Y(y|\boldsymbol{\theta}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{1}{2\sigma^2}(y-\mu)^2)$ with parameters $\boldsymbol{\theta} = [\mu, \sigma^2]^\mathsf{T}$
  - This pdf is a "bell curve" centered at μ with width σ:



Maximum-likelihood (ML) estimation

- Now say that we observe independent samples $\boldsymbol{y} = [y_1, \ldots, y_n]^{\mathsf{T}}$ of a random variable Y with pdf $p_Y(\cdot|\boldsymbol{\theta})$.
- Since we've assumed $\{y_i\}_{i=1}^n$ are independent & identically distributed (i.i.d.),

$$p(\boldsymbol{y}|\boldsymbol{\theta}) = \prod_{i=1}^{n} p_Y(y_i|\boldsymbol{\theta})$$

- For this fixed $\boldsymbol{y}$, the function $p(\boldsymbol{y}|\boldsymbol{\theta})$ versus $\boldsymbol{\theta}$ is called the "likelihood function"
- A common way of estimating $\boldsymbol{\theta}$ from observed $\boldsymbol{y}$ is to maximize the likelihood:

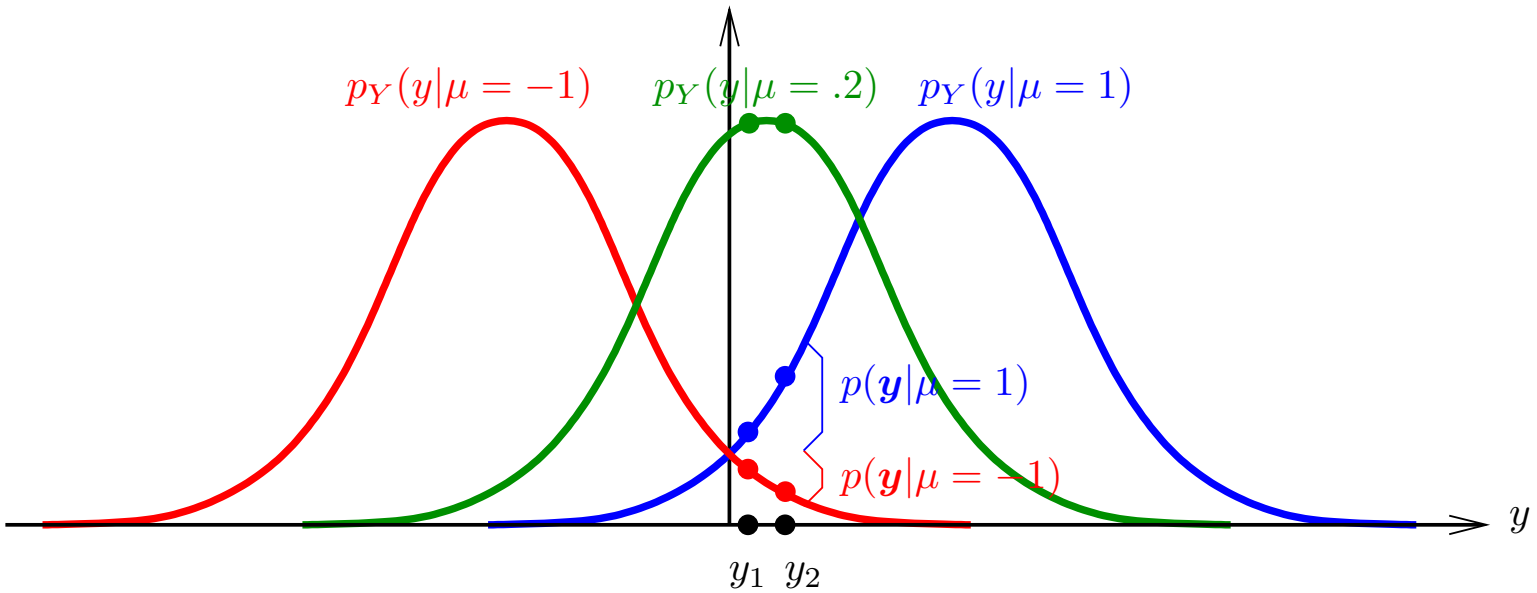
$$\boldsymbol{\theta}_{\text{ml}} \triangleq \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{y}|\boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \{\ln p(\boldsymbol{y}|\boldsymbol{\theta})\} = \arg\min_{\boldsymbol{\theta}} \{-\ln p(\boldsymbol{y}|\boldsymbol{\theta})\}$$

  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

Visualization of ML estimation

- Suppose we model $Y \sim \mathcal{N}(\mu, 1)$ with unknown mean μ (i.e., $\boldsymbol{\theta} = \mu$)
  - Thus $p_Y(y|\mu) = \frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}(y-\mu)^2)$
- We collect $n = 2$ samples $\boldsymbol{y} = [y_1, y_2]^{\mathsf{T}}$
- The likelihood equals $p(\boldsymbol{y}|\mu) = \prod_{i=1}^{n} p_Y(y_i|\mu) = \prod_{i=1}^{2} \frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}(y_i-\mu)^2)$
- We search for $\widehat{\mu}_{\mathsf{ml}}$, which is the $\mu \in \mathbb{R}$ that maximizes $p(\boldsymbol{y}|\mu)$
  - The drawing below implies that $\widehat{\mu}_{\mathsf{ml}} = 0.2$



ML estimation for linear regression

- Can we use ML estimation to fit the parameters $\boldsymbol{\beta}$ of linear regression? Yes!
- Suppose that our data obeys the linear-Gaussian model

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} \quad \text{with} \quad \{\epsilon_i\} \sim \text{i.i.d. } \mathcal{N}(0, \sigma^2)$$

  assuming standardized data, and thus $\boldsymbol{\beta} = [\beta_1, \ldots, \beta_d]^\mathsf{T}$ (i.e., no intercept β_0)
- In this case, $y_i = \boldsymbol{x}_i^\mathsf{T}\boldsymbol{\beta} + \epsilon_i$ and so $p(y_i|\boldsymbol{X}, \boldsymbol{\beta}) = \mathcal{N}(y_i; \boldsymbol{x}_i^\mathsf{T}\boldsymbol{\beta}, \sigma^2)$
- Since $\{\epsilon_i\}$ are independent, y_i are independent conditional on $\boldsymbol{X}, \boldsymbol{\beta}$, and so

$$p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\beta}) = \prod_{i=1}^n p(y_i|\boldsymbol{X}, \boldsymbol{\beta}) = \prod_{i=1}^n \frac{1}{\sqrt{2\pi\sigma^2}} \exp\Big(-\frac{1}{2\sigma^2}(y_i - \boldsymbol{x}_i^\mathsf{T}\boldsymbol{\beta})^2 \Big)$$

$$= \frac{1}{(2\pi\sigma^2)^{n/2}} \exp\Big(-\frac{1}{2\sigma^2} \sum_{i=1}^n (y_i - \boldsymbol{x}_i^\mathsf{T}\boldsymbol{\beta})^2 \Big) = \frac{\exp(-\frac{1}{2\sigma^2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2)}{(2\pi\sigma^2)^{n/2}}$$

- The ML estimate is then

$$\boldsymbol{\beta}_{\text{ml}} = \arg\min_{\boldsymbol{\beta}} \big\{ -\ln p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\beta}) \big\} = \arg\min_{\boldsymbol{\beta}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2 = \boldsymbol{\beta}_{\text{ls}}$$

- 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

$$\boldsymbol{\beta}_{\text{map}} \triangleq \arg\max_{\boldsymbol{\beta}} p(\boldsymbol{\beta}|\boldsymbol{y}, \boldsymbol{X}),$$

  the most probable $\boldsymbol{\beta}$ given the data $(\boldsymbol{X}, \boldsymbol{y})$

- Bayes rule says

$$p(\boldsymbol{\beta}|\boldsymbol{y}, \boldsymbol{X}) = \frac{p(\boldsymbol{y}|\boldsymbol{\beta}, \boldsymbol{X})p(\boldsymbol{\beta}|\boldsymbol{X})}{p(\boldsymbol{y}|\boldsymbol{X})} = \frac{p(\boldsymbol{y}|\boldsymbol{\beta}, \boldsymbol{X})p(\boldsymbol{\beta})}{p(\boldsymbol{y}|\boldsymbol{X})}$$

  where $p(\boldsymbol{\beta})$ models prior belief about $\boldsymbol{\beta}$

$$\Rightarrow \quad \boldsymbol{\beta}_{\text{map}} = \arg\max_{\boldsymbol{\beta}} \frac{p(\boldsymbol{y}|\boldsymbol{\beta}, \boldsymbol{X})p(\boldsymbol{\beta})}{p(\boldsymbol{y}|\boldsymbol{X})} = \arg\max_{\boldsymbol{\beta}} \left\{p(\boldsymbol{y}|\boldsymbol{\beta}, \boldsymbol{X})p(\boldsymbol{\beta})\right\}$$

- Interpretations:

  likelihood $p(\boldsymbol{y}|\boldsymbol{\beta}, \boldsymbol{X})$: 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

$$\begin{aligned}\boldsymbol{\beta}_{\text{map}} &= \arg\min_{\boldsymbol{\beta}} \left\{ \tfrac{1}{2\sigma^2} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2 - \ln p(\boldsymbol{\beta}) \right\} \\ &= \arg\min_{\boldsymbol{\beta}} \Big\{ \underbrace{\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2}_{\text{RSS}(\boldsymbol{\beta})} \underbrace{-2\sigma^2 \ln p(\boldsymbol{\beta})}_{+\,\phi(\boldsymbol{\beta})} \Big\}\end{aligned}$$

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

MAP interpretations of Ridge Regression and LASSO

- So we have

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

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

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

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

Summary of statistical estimation

- Supervised machine learning can be framed as statistical parameter estimation.
- To do this, a probablistic model $p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta})$ is posed that relates the training features $\boldsymbol{X}$ and model parameters $\boldsymbol{\theta}$ to the training data $\boldsymbol{y}$
- The maximum-likelihood (ML) estimate of $\boldsymbol{\theta}$ is

$$\boldsymbol{\theta}_{\text{ml}} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \big\{ -\ln p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta}) \big\},$$

  which employs no prior belief about $\boldsymbol{\theta}$

- The maximum a posteriori (MAP) estimate of $\boldsymbol{\theta}$ is

$$\boldsymbol{\theta}_{\text{map}} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\boldsymbol{y},\boldsymbol{X}) = \arg\min_{\boldsymbol{\theta}} \big\{ -\ln p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta}) - \ln p(\boldsymbol{\theta}) \big\},$$

  which employs the prior belief $p(\boldsymbol{\theta})$

Outline

- Motivating Example: Predicting Prostate Cancer
- Feature Selection
- Ridge Regression and LASSO
- Maximum Likelihood and MAP
- Regression with Vector-Valued Targets

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 β_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 $i = 1, \ldots, n$, we get the model

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

Regularized regression with vector-valued targets

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

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

- 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
  - The standard matrix norm $\|\boldsymbol{B}\| = \|\boldsymbol{B}\|_2$ is not useful here!

- 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 work with vector-valued targets

Learning objectives

- Understand motivation for and concept of feature selection
- Understand feature selection methods based on:
  - exhaustive search
  - stepwise selection
  - univariate statistics
  - 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