# **Homework 2:** Priors, regularization, and shrinkage

STATS348, UChicago, Spring 2024

---

## **Your name here:** Daniel Li

Open in Colab

---

## Instructions

The purpose of this homework is apply the ideas from lectures 3 and 4 on regularizers, priors, and shrinkage.

Please fill out your answers in the provided spaces below. When you are finished, export the notebook as a PDF, making sure that all of your solutions are clearly visible.

Assignment is due **Saturday April 6, 11:59pm** on GradeScope.

---

## **Problem 1:** Regularization and Priors

Consider a standard regression setting with fixed design $X \in \mathbb{R}^{n \times p}$ and

$$y = X\beta + \varepsilon$$

where $\varepsilon \sim \mathcal{N}(0, \sigma^2 I_p)$ for variance $\sigma^2$ considered known and fixed.

The **elastic net criterion** is a regularized loss function defined as $\ell_{\text{EN}}(\lambda_1, \lambda_2, \beta) = ||y - X\beta||_2^2 + \lambda_1 ||\beta||_1 + \lambda_2 ||\beta||_2^2$, and the **elastic net estimator** is defined as its minimizer:

$$\widehat{\beta}^{\text{EN}}(\lambda_1, \lambda_2) := \underset{\beta}{\text{argmin}}\ \ell_{\text{EN}}(\lambda_1, \lambda_2, \beta)$$

In lecture, we saw a correspondence between Ridge and LASSO estimators with MAP estimates under normal and Laplace priors, respectively.

---

- **1a)** Provide the form **up to proportionality** of a prior density $P(\beta)$ on the regression coefficients such that the MAP estimate under $P(\beta)$ corresponds to $\widehat{\beta}^{\text{EN}}(\lambda_1, \lambda_2)$. In the space below, please state $P(\beta)$ clearly, and provide a brief justification.

$$P(\beta) \propto_\beta \exp(\lambda_1 |||\beta|||_1 + \lambda_2 |||\beta|||_2^2)$$

Elastic net is just a combination of LASSO and Ridge

---

- **1b)** Consider a dataset where the input feature $x$ is generated from a uniform distribution over the interval $[-3, 3]$, and the corresponding target $y$ is generated as

$$y_i = 2x_i^3 - x_i^2 + \varepsilon_i$$

where $\varepsilon_i \sim \mathcal{N}(0, 10)$ is a noise term (with *variance* 10). Use the code block below to simulate a dataset of $n = 20$ data points in Python.

In [ ]:
```python
# Your code here
import numpy as np
import math

x = np.random.uniform(-3, 3, 20)
y = 2 * x**3 - x**2 + np.random.normal(0, math.sqrt(10), 20)
y
```

Out[ ]:
```
array([-26.21739859, -16.33355155, -43.17076021, -18.50811715,
        -0.564598  , -14.89856019,   2.33980075,  44.28937676,
         7.08590584, -13.08142731,   6.56091283, -21.65597249,
        24.16784869,   4.18538248, -41.27477737,   1.24360274,
        -4.60991902,   0.40248132,  15.30786364,   3.72648863])
```

---

Now consider the following model of the data using a 5th degree polynomial regression.

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \ldots + \beta_5 x_i^5 + \varepsilon_i$$

where the noise is assumed $\varepsilon_i \sim \mathcal{N}(0, 1)$ (with *variance* 1).

- **1c)** Estimate the coefficients using maximum likelihood as $\widehat{\beta}^{\mathrm{MLE}}$. In the code block below, do this programmatically using scikit-learn's PolynomialFeatures preprocessor. (You may have to import other methods/libraries as well.)

```
In [ ]: from sklearn.preprocessing import PolynomialFeatures
        import scipy.optimize as opt
        import scipy.stats

        # Your code here
        poly = PolynomialFeatures(5)
        X = poly.fit_transform(x.reshape(-1, 1))


        def likelihood(beta, X, y, std=1):
            x = y - X @ beta
            return np.sum(scipy.stats.norm.logpdf(x, 0, std))


        def estimate_beta(X, y):
            return opt.minimize(
                lambda beta: -likelihood(beta, X, y), np.zeros(X.shape[1]), method="L-BFGS-B"
            ).x


        beta_mle = estimate_beta(X, y)
        beta_mle
```

```
Out[ ]: array([ 3.33931461,  0.03298756, -3.23081042,  1.59389113,  0.23149808,
                0.03900222])
```

```
In [ ]: np.linalg.inv(X.T @ X) @ X.T @ y
```

```
Out[ ]: array([ 3.33931992,  0.03300063, -3.23081385,  1.59388318,  0.23149824,
                0.03900306])
```

---

Now, consider a prior of $\beta \sim \mathcal{N}(0, \sigma_0^2)$ on the regression coefficients.

- **1d)**: Provide the form of the MAP solution $\widehat{\beta}^{\mathrm{MAP}}(\sigma_0^2)$ below:

$$\widehat{\beta}^{\text{MAP}}(\sigma_0^2) = y^T x(\frac{1}{\sigma_0^2} + x^T x)^{-1}$$

---

- **1e)** In the code block below, implement the MAP estimator for a given $\sigma_0^2$ (using any methods or libraries you like):

In [ ]:
```python
def map_estimate(y, X, sigma0):
    # Your code here
    return np.linalg.inv(X.T @ X + 1 / sigma0**2 * np.eye(X.shape[1])) @ X.T @ y
```

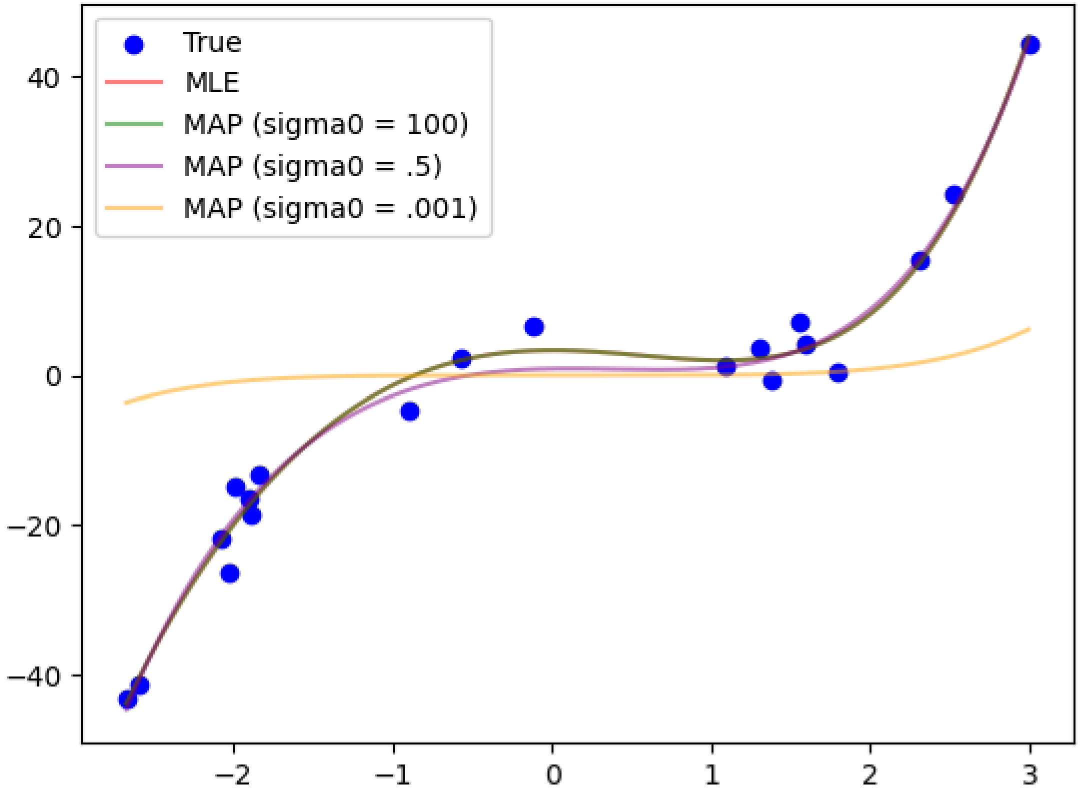
---

- **1f)** Fit the MAP estimate to the synthetic data you generated above, experimenting with different values for $\sigma_0^2$. Find a value that seems "too large", a value that seems "too small", and a value that seems "just right" based on plotting and inspecting the learned regression functions. (These judgements are subjective, we are not expecting specific values, just reasonable ones.) Once you have selected three values, display clearly in a single plot the following:

  - The dataset $(x_1, y_1), \ldots, (x_{20}, y_{20})$
  - The estimated regression function using using $\widehat{\beta}_{\text{MLE}}$
  - The three estimated regression functions $\widehat{\beta}^{\text{MAP}}(\sigma_0^2)$ for the selected three values of $\sigma_0^2$

  Make sure your plot includes a legend that clearly indicates the different functions.

In [ ]:
```python
import matplotlib.pyplot as plt
import seaborn as sns

beta_map1 = map_estimate(y, X, 1000)
beta_map2 = map_estimate(y, X, 0.5)
beta_map3 = map_estimate(y, X, 0.001)

x_lin = np.linspace(min(x), max(x), 1000)
X_lin = poly.fit_transform(x_lin.reshape(-1, 1))

# Your code here
fig = plt.figure()
plt.scatter(x, y, color="blue", label="True")
plt.plot(x_lin, X_lin @ beta_mle, label="MLE", color="red", alpha=0.5)
plt.plot(x_lin, X_lin @ beta_map1, label="MAP (sigma0 = 100)", color="green", alpha=0.5)
plt.plot(x_lin, X_lin @ beta_map2, label="MAP (sigma0 = .5)", color="purple", alpha=0.5)
plt.plot(
    x_lin, X_lin @ beta_map3, label="MAP (sigma0 = .001)", color="orange", alpha=0.5
)
plt.legend()
plt.show()
```



---

- **1g)** Discuss your findings plot above.
  - How does the MLE compare to the three different MAP solutions? MLE is close to the high sigma MAP because of the inverse relationship with the sigma_0. Overall, it does pretty well.
  - Why did you select the three values of $\sigma_0^2$ that you did? I chose one to over regularize, one that essentially didn't do anything, and one in the middle

## **Problem 2:** Estimating SEPs with binomial trials

### Setting

It is May 1968 and the USS *Scorpion* has just disappeared somewhere in the Atlantic Ocean, likely off the coast of Spain. You are the lone statistician on board the USS *Mizar*, which has been dispatched to find the missing submarine. Your job is to guide the search as best you can, given the data at your disposal.

### Search effectiveness probability (SEP)

As we saw in lecture, an important component of our decision problem is the *search effectiveness probability* of each search cell $k$

$$q_k = P(\text{finding the sub in } k \mid \text{sub is in } k \text{ and the divers search } k) \tag{1}$$

### Binomial trials

To collect data that we can use to estimate these SEPs, our divers have been running trials to see if they can recover large objects thrown overboard in each cell $k$. Define the following quantity:

$$y_k \in \{0, \ldots, n_k\} \qquad \text{the number of successful trials in } k \tag{2}$$

where $n_k$ is the total number of trials in $k$. We will assume the following likelihood for $y_k$ given the SEP $q_k$:

$$y_k \overset{\text{ind.}}{\sim} \text{Binom}(n_k,\, q_k) \text{ for cell } k = 1 \ldots K \tag{3}$$

### Beta prior

Now further assume the following prior for the SEPs:

$$q_k \overset{\text{iid}}{\sim} \text{Beta}(a_0,\, b_0) \text{ for cell } k = 1 \ldots K \tag{4}$$

where $a_0$ and $b_0$ are the Beta prior's shape parameters.

---

- **2a):** Provide an analytic expression (i.e., without any integrals) for the negative log marginal likelihood $-\log P(\boldsymbol{y}_{1:K} \mid \boldsymbol{n}_{1:K}, a_0,\, b_0)$ where $\boldsymbol{y}_{1:K} \equiv (y_1, \ldots, y_K)$ and $\boldsymbol{n}_{1:K} \equiv (n_1, \ldots, n_K)$ are the data across all cells. Provide your answer below, along with a brief justification.

$$-\log P(\boldsymbol{y}_{1:K} \mid \boldsymbol{n}_{1:K}, a_0, b_0\,) = -\sum_{k=1}^{K}\left[\log\binom{n_k}{y_k} + \log B(y_k + a_0, n_k - y_k + b_0) - \log B(a_0, b_0)\right]$$

Where B is the pdf of the beta. When you integrate out $q_k$ you recover a beta distribution

---

- **2b):** In the code cell below, implement a function that takes in two arrays for $\boldsymbol{y}_{1:K}$ and $\boldsymbol{n}_{1:K}$, along with a value of the parameters $(a_0, b_0)$, and computes the negative marginal log likelihood. We recommend you use functions in the `numpy` and/or `scipy` Python libraries, but you may use any other libraries if you like.

In [ ]:
```
import numpy as np
import scipy.stats  # for stats-related methods
import scipy.special  # for special functions (e.g., gammaln)
from scipy.special import betaln
from scipy.special import comb


def neg_log_marginal_likelihood(params, y, n):
    """Calculate the negative log-marginal likelihood for the beta-binomial model.

    Args:
        params (tuple): the parameters of the marginal likelihood (a0, b0)
        y (array): the number of successes for each trial
        n (array): the number of trials

    Returns:
        float: the negative log-marginal likelihood
    """
    a0, b0 = params

    # Your code here

    return -np.sum(np.log(comb(n, y)) + betaln(y + a0, n - y + b0) - betaln(a0, b0))
```

---

- **2c):** Now, in the code cell below, implement a method for fitting the parameters $(a_0, b_0)$ which relies on your implementation of the negative log marginal likelihood. We recommend using `scipy.optimize.minimize` (make sure to read documentation and to experiment with different settings). You are allowed to use other methods and libraries, if you so choose.

In [ ]:
```python
def fit_marginal_likelihood(y, n):
    """Fit the parameters of the marginal likelihood to the data.

    Args:
        y (array): the number of successes for each trial
        n (array): the number of trials

        If your function requires other input arguments, include a description here.

    Returns:
        tuple: the MLE for a0 and b0
    """

    # Your code here
    a0_mle, b0_mle = opt.minimize(
        lambda params: neg_log_marginal_likelihood(params, y, n),
        np.array([1, 1]),
        method="L-BFGS-B",
    ).x

    return a0_mle, b0_mle
```

---

- **2d):** Use your method to fit $a_0$ and $b_0$ to the trial data. In the cell below, load in the trial data and call your method.

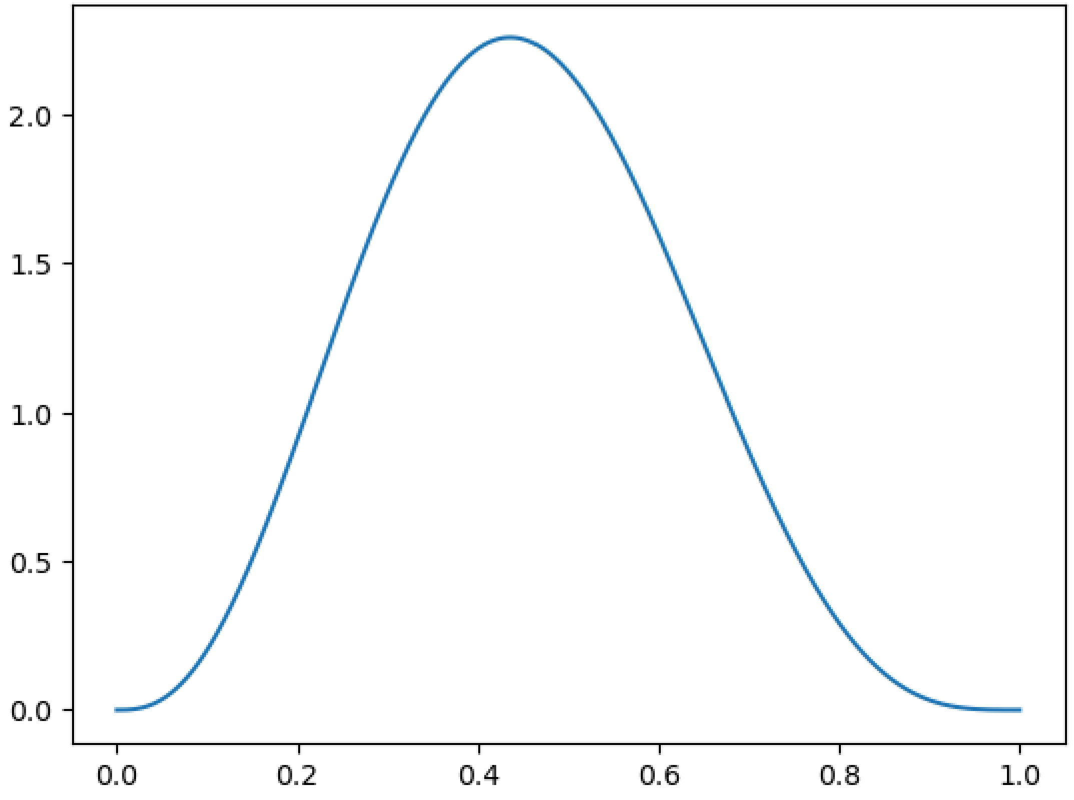
In [ ]:
```python
# load in the beta-binomial trial data
import pandas as pd

df_trials = pd.read_csv("binomial_trials.csv")
y = df_trials["n_successes"].values
n = df_trials["n_trials"].values

# if your fit_marginal_likelihood function takes extra args, modify this code to pass them in here.
a0_mle, b0_mle = fit_marginal_likelihood(y, n)

print(a0_mle, b0_mle)
```

```
3.7857845729701296 4.60463591368849
```

---

- **2e):** This is an empirical Bayes procedure that can be loosely understood as "fitting the prior". In the code cell below, create a plot that visualizes the the PDF of the "fitted" Beta prior---i.e., $\mathrm{Beta}(q;\, \widehat{a}_0\, \hat{b}_0)$.

```
In [ ]: # Your code here
        fig = plt.figure()
        t = np.linspace(0, 1, 1000)
        ft = scipy.stats.beta.pdf(t, a0_mle, b0_mle)
        plt.plot(t, ft)
        plt.show()
```



---

- **2f)**: In the code cell below, use your fitted parameters $(\widehat{a}_0, \hat{b}_0)$ to compute the posterior means of all $K$ SEPs

$$\hat{q}_k^{\text{post-mean}} = \mathbb{E}[q_k \mid y_k, n_k, \widehat{a}_0, \hat{b}_0]$$

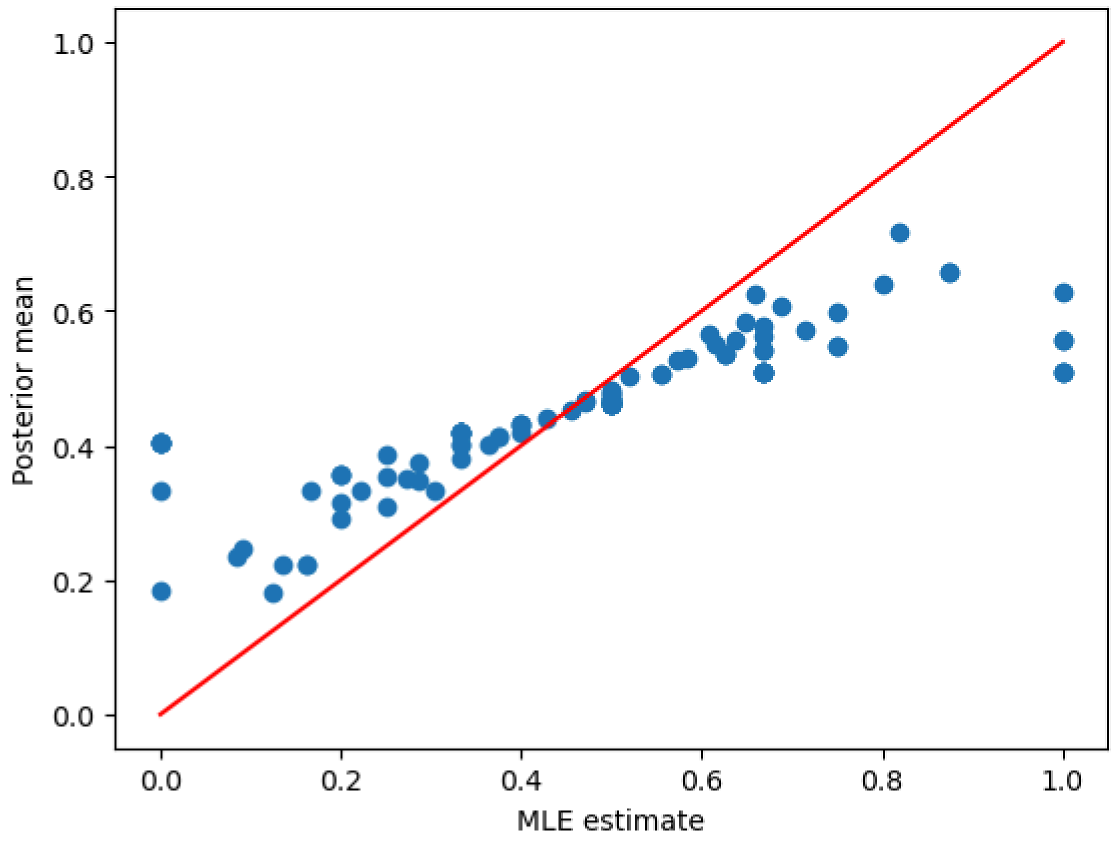
and compare them to the maximum likelihood estimates

$$\hat{q}_k^{\text{MLE}} = \underset{q_k}{\operatorname{argmax}}\, P(y_k \mid n_k, q_k)$$

. More specifically:

- Compute the posterior means.
- Compute the maximum likelihood estimates.
- Generate a scatter plot where each $(x, y)$ point is a pair $(\hat{q}_k^{\text{MLE}}, \hat{q}_k^{\text{post-mean}})$ for all $k = 1 \ldots K$. For reference, also include the line $x = y$, and make sure the $x$- and $y$-axis both range over the full set of possible values.

In [ ]:
```
# Your code here
post_means = (y + a0_mle) / (n + a0_mle + b0_mle)
mle_est = y / n
fig = plt.figure()
plt.scatter(mle_est, post_means)
plt.xlabel("MLE estimate")
plt.ylabel("Posterior mean")
plt.plot([0, 1], [0, 1], color="red")
plt.show()
```



---

- **2f)**: Discuss the plot you just generated.
    - What is the relationship between the maximum likelihood estimates and the posterior means? While they are positively correlated the there is a shallower slope than the identity line. So away from 0.5 posterior mean is greater on the left and lower on teh right.
    - Why does this make sense based on your understanding of the procedure you have implemented? This makes sense because you are updating your knowledge in the posterior means, so with more information, you probably won't be as extremly opiniated on if something is located in the grid.
    - Comment on any other observations or insights. There seems to be a shrinkage in the band of the range closer to 0.5.