INF367A Exercise 9

Naphat Amundsen

March 14, 2020

Introduction

This exercise is about sampling methods

Context: Earlier, we have performed Bayesian linear regression assuming that the noise precision β is known. Let us now formulate a full Bayesian linear regression model by placing a prior on β.
Assume that we have observed n pairs (x_i, y_i) where x_i are the feature values and y_i is the label. Let $w \in \mathbb{R}^d$ be our regression weights. The likelihood is

$$P(y|x, w, \beta) = \prod_{i=1}^{n} N(y_i|w^T x_i, \beta^{-1}).$$

As earlier, we place Gaussian prior on the regression weights:

$$P(w) = N(w|0, \alpha^{-1} I)$$

where α (prior precision) is a user-defined hyperparameter. For β, we use a Gamma prior

$$P(\beta) = \text{Gamma}(\beta|a_0, b_0)$$

where a_0 and b_0 are user-defined hyperparameters.

The goal is to compute the posterior

$$P(w, \beta|x, y)$$

This distribution does not have a closed-form solution and therefore we use sampling to approximate the posterior.

1 Gibbs sampling

1.1 Derive conditional distributions

Derive a Gibbs sampler for the above model. That is, derive the full conditional distributions $P(w|x, y, \beta)$ and $P(\beta|x, y, w)$.
Hint: When deriving the conditional distribution for w or β, you can drop all terms that do not depend on w or β, respectively. Note that both distributions involve conjugate priors.

Case for w

Let $a = a_0,\ b = b_0$

$$P(w|y,x,\beta) \propto P(\beta)P(w)P(y|x,w,\beta) \tag{1}$$
$$= \text{Gamma}(\beta|a,b)\cdot N(w|0,\alpha I)\cdot \prod_{i=1}^{n} N(y_i - w^T x_i) \tag{2}$$
$$= \frac{1}{\Gamma(a)} b^a \beta^{a-1} e^{-b\beta} e^{-\frac{1}{2}w^T\alpha I w} \prod_{i=1}^{n} e^{-\frac{1}{2}(y_i - w^T x_i)^T \beta (y_i - w^T x_i)} \tag{3}$$
$$\propto \exp\left[-\frac{1}{2}w^T\alpha I w + \sum_{i=1}^{n} -\frac{1}{2}(y_i - w^T x_i)^T\beta(y_i - w^T x_i)\right] \tag{4}$$

Logarithm the thing to make math doable

$$\Rightarrow \ln P(w|y,x,\beta) = \ln\left(\exp\left[-\frac{1}{2}w^T\alpha I w + \sum_{i=1}^{n} -\frac{1}{2}(y_i - w^T x_i)^T\beta(y_i - w^T x_i)\right]\right) \tag{5}$$
$$= -\frac{1}{2}w^T\alpha I w - \frac{1}{2}\sum_{i=1}^{n}(y_i - w^T x_i)^T\beta(y_i - w^T x_i) \tag{6}$$
$$= -\frac{1}{2}\left[w^T\alpha I w + \sum_{i=1}^{n}(y_i - w^T x_i)^T\beta(y_i - w^T x_i)\right] \tag{7}$$

Now want to write the thing in quadratic form:

$$\frac{1}{2}\theta^T A\theta + b^T\theta \tag{8}$$

Let $w^T x_i = p$ (don't want to use $\hat{y}_i$ since it confuses me when there are a lot of y_i symbols as well)

$$= -\frac{1}{2}\left[w^T\alpha I w + \sum_{i=1}^{n}(y_i - p_i)^T\beta(y_i - p_i)\right] \tag{9}$$

$$= -\frac{1}{2}\left[w^T\alpha I w + \sum_{i=1}^{n} y_i^T\beta y_i - y_i^T\beta p_i - p_i^T\beta y_i + p_i^T\beta p_i\right] \tag{10}$$

$$\propto -\frac{1}{2}\left[w^T\alpha I w + \sum_{i=1}^{n} \overbrace{-y_i^T\beta p_i - p_i^T\beta y_i + p_i^T\beta p_i}^{\beta \text{ is scalar}}\right] \tag{11}$$

$$= -\frac{1}{2}\left[w^T\alpha I w + \sum_{i=1}^{n} -y_i^T\beta p_i - y_i^T\beta p_i + p_i^T\beta p_i\right] \tag{12}$$

$$= -\frac{1}{2}\left[w^T\alpha I w + \sum_{i=1}^{n} -2y_i^T\beta p_i + p_i^T\beta p_i\right] \tag{13}$$

$$= -\frac{1}{2}\left[w^T\alpha I w + \sum_{i=1}^{n} -2y_i^T\beta w^T x_i + \beta w^T x_i x_i^T w\right] \tag{14}$$

$$= -\frac{1}{2}\left[w^T\alpha I w - 2\sum_{i=1}^{n} y_i^T\beta x_i^T w + \sum_{i=1}^{n} \beta w^T x_i x_i^T w\right] \tag{15}$$

$$= -\frac{1}{2}\left[w^T\left(\alpha I + \beta\sum_{i=1}^{n} x_i x_i^T\right) w - 2\left(\beta\sum_{i=1}^{n} y_i^T x_i^T\right) w\right] \tag{16}$$

$$= -\frac{1}{2}w^T\left(\alpha I + \beta\sum_{i=1}^{n} x_i x_i^T\right) w + \left(\beta\sum_{i=1}^{n} y_i^T x_i^T\right) w \tag{17}$$

Now it is in quadratic form yo, let

$$A = \left(\alpha I + \beta\sum_{i=1}^{n} x_i x_i^T\right), \quad b = \left(\beta\sum_{i=1}^{n} y_i x_i\right) \tag{18}$$

Then by completing the square, we obtain the parameters S (variance), m (mean) for a Gaussian (see lecture slides).

$$S = A^{-1} = \left(\alpha I + \beta\sum_{i=1}^{n} x_i x_i^T\right)^{-1} \tag{19}$$

$$m = A^{-1}b = \beta S\sum_{i=1}^{n} y_i x_i \tag{20}$$

$$\underline{\underline{\Rightarrow P(w|y,x,\beta) = \mathcal{N}(w|m,S)}} \tag{21}$$

Case for β

Let $a = a_0,\ b = b_0$

$$P(b|x,y,w) \propto P(\beta)P(w)P(y|x,w,\beta) \tag{22}$$

$$= \text{Gamma}(\beta|a,b)\cdot N(w|0,\alpha I)\cdot\prod_{i=1}^{n} N(y_i - w^T x_i) \tag{23}$$

$$= \frac{1}{\Gamma(a)} b^a \beta^{a-1} e^{-b\beta} e^{-\frac{1}{2}w^T \alpha I w} \prod_{i=1}^{n} e^{-\frac{1}{2}(y_i - w^T x_i)^T \beta (y_i - w^T x_i)} \tag{24}$$

$$\propto \beta^{a-1} e^{-b\beta} \prod_{i=1}^{n} e^{-\frac{1}{2}(y_i - w^T x_i)^T \beta (y_i - w^T x_i)} \tag{25}$$

$$= \beta^{a-1} \exp\left[-b\beta + \sum_{i=1}^{n} -\frac{1}{2}(y_i - w^T x_i)^T \beta (y_i - w^T x_i)\right] \tag{26}$$

$$= \beta^{a-1} \exp\left[-\beta\left(b + \frac{1}{2}\sum_{i=1}^{n}(y_i - w^T x_i)^T (y_i - w^T x_i)\right)\right] \tag{27}$$

$$\propto \underline{\underline{\text{Gamma}(a, b + \frac{1}{2}\sum_{i=1}^{n}(y_i - w^T x)^T (y_i - w^T x_i))}} \tag{28}$$

1.2 Implement the Gibbs sampler

Implement the Gibbs sampler. Download the data new_york_bicycles.csv. The first column is the number of daily riders on Brooklyn Bridge (x) and the second column is the number of daily riders on Manhattan Bridge (y). Use your Gibbs sampler to learn a simple linear model (intercept and slope) for this data.

Compute the logarithm of the joint probability of the parameter values (which is proportional to the posterior).

$$\log P(y, x, w^t, \beta^t) = \log P(\beta^t) + \log P(w^t) + \sum_{i=1}^{n} \log P(y_i | x_i, w^{(t)}, (\beta^{(t)})^{-1})$$

after each iteration. Plot a trace plot of these values (x-axis is the iteration number). What can you say about the convergence based on this plot? Plot trace plots for the parameter values w_0 , w_1 and β. What can you say about convergence and mixing based on these plots. Plot the marginal distributions of the parameters w_0 , $w1$ and β. Remember to exclude burn-in.

Hint: Remember that Gamma(a, b) is gamma(a=a, scale=1/b) in Scipy.
Hint: You want sample thousands of samples. Typically, burn-in is about half of the samples.

Idea

1. Initialize values for w_0^0, w_1^0, β
2. For index t in n iterations do:
   - 2.1. Sample (w_0^{t+1}, w_1^{t+1}) from $\mathcal{N}\left(w^t|m^t, S^t\right)$ (m and S are defined in (20) and (19))
   - 2.2. Sample β^{t+1} from Gamma $\left(a, b + \frac{1}{2}\sum_{i=1}^{n}(y_i - {w^t}^T x)^T(y_i - {w^t}^T x_i)\right)$
3. Plot the posterior (or something proportional to it) for each iteration.
4. Plot the parameters for each iteration.

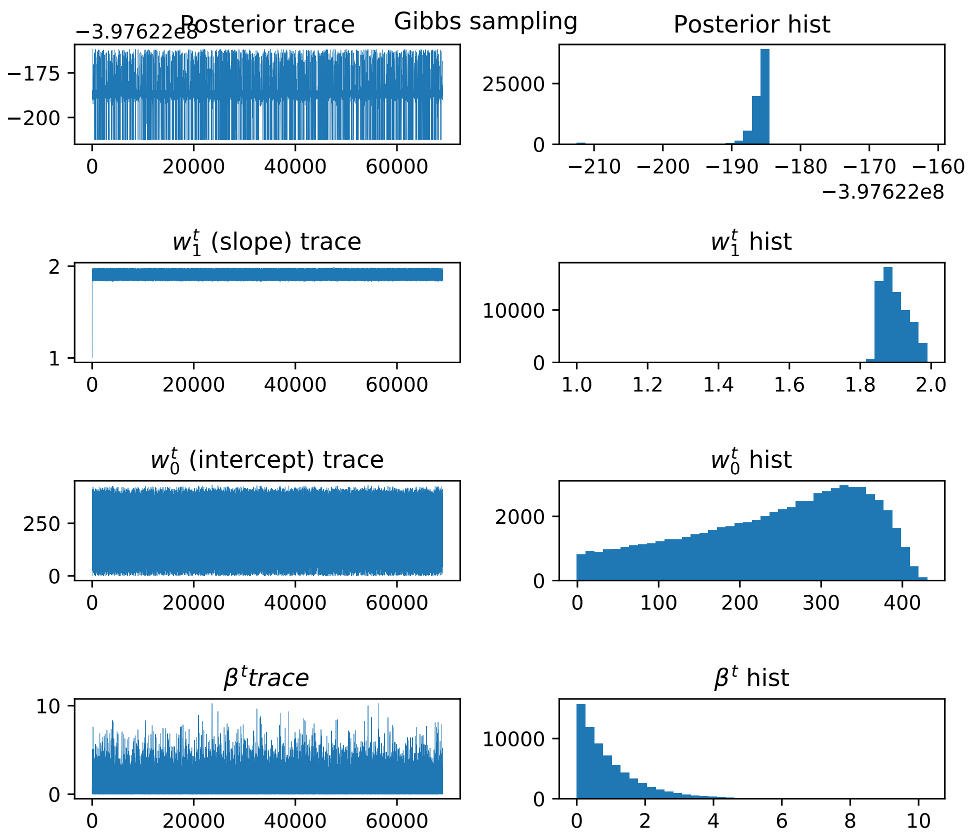

Figure 1: Just looking at the trace plots does not give me very much, but the histograms looks nice and looks reasonable.

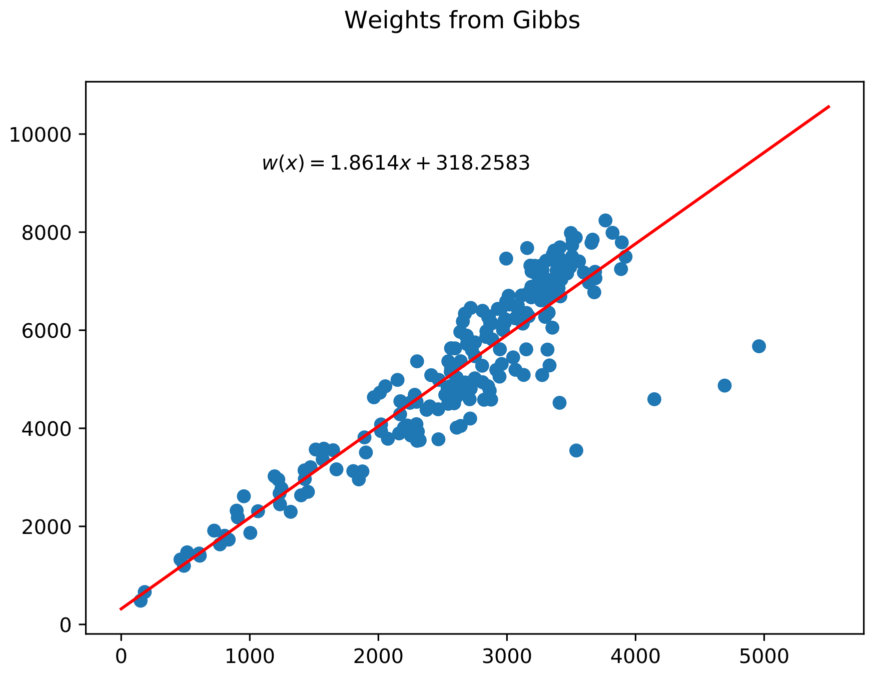

Figure 2: Taking the argmax of the regression weight histograms gives a reasonable trendline.

2 Metropolis(-Hastings) Algorithm

Derive a Metropolis(-Hastings) algorithm. That is, come up with a proposal distribution $q(w^*, \beta^* | w, \beta)$. You should probably add a parameter or two controlling the ”step-size”, that is, how far from the current values can the proposed values be.

Use multivariate normal distribution with diagonal covariance matrix since I like it (and it is radially symmetric when there is no covariance, which is nice). Algorithm is then:

```
1. Initialize $\theta = (w_0^0, w_1^0, \beta^0)$
2. for index $t$ in iterations:
    2.1. Sample $\theta^*$ from $\mathcal{N}(\theta^{t-1}, \Sigma)$
    2.2. Let $A = \min\left\{1, \frac{P(\theta^*|D)}{P(\theta^{t-1}|D)}\right\}$
    2.3. Sample $u \sim \text{Uniform}(0, 1)$
    2.4. If $u < A$, set $\theta^t = \theta^*$, else $\theta^t = \theta^{t-1}$
```

Implement the Metropolis-Hastings algorithm and do the same steps as in section 1.2. In addition, keep track of the proportion of proposal that were actually accepted.

Hint: To avoid numerical problems in computing the acceptance ratio, one can use the following formula:

$$\frac{P(k^*)}{P(k)} = e^{\log P(k^*) - \log P(k)}.$$

Note that here you need the logarithm of the joint probability which you implemented already in task 2.

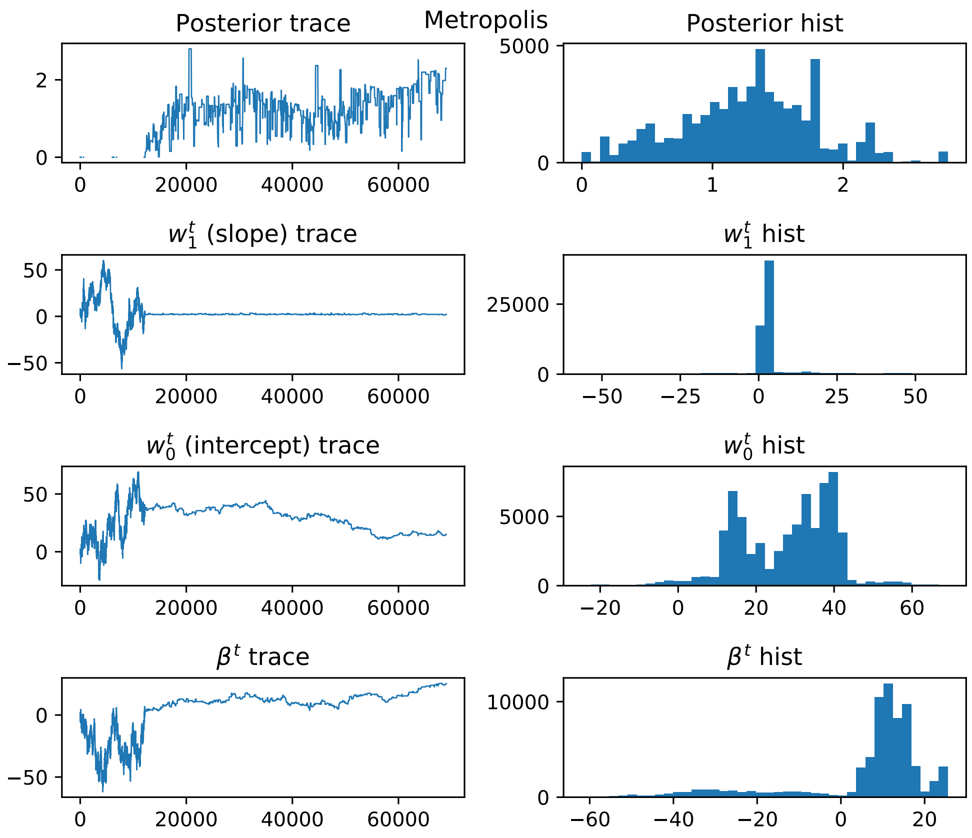

Figure 3: The burnout time seems to end at 20000 iterations according to my eyes. Afterwards we see that the parameter values stay relatively flat opposed to movement during the burnout. The histograms aren't as nice as Gibbs though, but they include the burnout values.

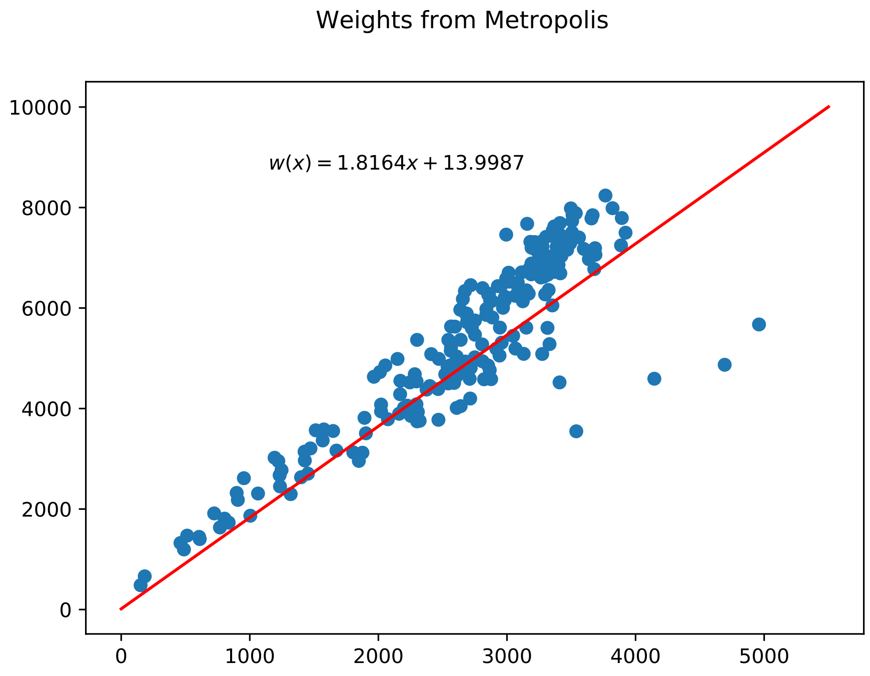

Figure 4: After cutting the first 20 000 samples (the burnout), then doing argmax on the histograms of the samples linear regression parameters resulted in a reasonable regression line as well.

3 Comparison of the methods

Both works, Gibbs works better (better line i'd say) and is much faster and does not require any parameter optimization. It is expected that Gibbs would perform better as it samples from conjugate priors, opposed to Metropolis which does some kind of drunken search for the likelihood extrema. It is kind of funny though so I'll give it that.

A Gibbs code

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from scipy.stats import multivariate_normal, gamma, norm, rv_histogram
np.random.seed(42069)

class LinRegGibbs:
    def __init__(self, alpha=10, beta=0.1, a=1, b=1e-2):
        self.alpha = alpha
        self.beta = beta
        self.a = a
        self.b = b
    def fit(self, X, y, n=69000):
        self.n = n
        self.X = X
        self.y = y
        self.ws = np.empty((n, 2))
        self.betas = np.empty(n)

        I = np.eye(2)
        self.ws[0] = np.array([1,1])
        self.betas[0] = self.beta
        origo = np.zeros(2)
        for i in range(1, n):
            S = np.linalg.inv(I*self.alpha + self.betas[i-1]*X.T@X)
            m = self.betas[i-1]*S@(X*y.reshape(-1,1)).sum(0)
            self.ws[i] = multivariate_normal.rvs(mean=m, cov=S, size=1)

            p = y - self.ws[i]@X.T
            self.betas[i] = gamma.rvs(self.a,1/(self.b + 0.5 * p.T@p))

    def posterior(self):
        epsilon = 1e-12
        I = np.eye(2)
        origo = np.zeros(2)
        logw = np.log(multivariate_normal.pdf(self.ws, origo, self.alpha*I
                                             ).clip(epsilon))
        logbeta = np.log(gamma.pdf(self.betas, self.a, self.b).clip(
                                             epsilon))
        logdata =\
            np.log(norm.pdf(
                x=self.y-self.ws@X.T,
                loc=np.zeros((self.n,1)),
                scale=1/self.betas.reshape(-1,1)
            ).clip(epsilon)).sum()
        self.logposterior = logw+logbeta+logdata

    def plot_result(self):
        self.posterior()
        fig, axes = plt.subplots(4,2, figsize=(7,6))
        ax_left = axes[:,0].ravel()
        ax_right = axes[:,1].ravel()
        ax_left[0].plot(self.logposterior, linewidth=0.2)
        ax_left[0].set_title(r'Posterior trace')
        ax_left[1].plot(self.ws[:,0], linewidth=0.2)
        ax_left[1].set_title(r'$w_1^t$ (slope) trace')
        ax_left[2].plot(self.ws[:,1], linewidth=0.2)
        ax_left[2].set_title(r'$w_0^t$ (intercept) trace')
        ax_left[3].plot(self.betas, linewidth=0.2)
        ax_left[3].set_title(r'$\beta^t trace$')

        ax_right[0].hist(self.logposterior, bins=40)
        ax_right[0].set_title(r'Posterior hist')
        ax_right[1].hist(self.ws[:,0], bins=40)
        ax_right[1].set_title(r'$w_1^t$ hist')
        ax_right[2].hist(self.ws[:,1], bins=40)
        ax_right[2].set_title(r'$w_0^t$ hist')
        ax_right[3].hist(self.betas, bins=40)
        ax_right[3].set_title(r'$\beta^t$ hist')
        fig.tight_layout()
        fig.suptitle('Gibbs sampling')
        plt.savefig('gibbs.pdf')
        plt.show()

        xrange = np.linspace(1.5,2.5,1000)
        slope_hist = rv_histogram(np.histogram(self.ws[:,0], bins=100))
        slope = xrange[slope_hist.pdf(xrange).argmax()]

        xrange = np.linspace(200,420,1000)
        intercept_hist = rv_histogram(np.histogram(self.ws[:,1], bins=69))
        intercept = xrange[intercept_hist.pdf(xrange).argmax()]
        return intercept, slope

if __name__ == '__main__':
    df = pd.read_csv('new_york_bicycles2.csv')
    X_whole = df.values
    X = np.column_stack([X_whole[:,0], np.ones(len(X_whole))])
    y = X_whole[:,1]

    lol = LinRegGibbs()
    lol.fit(X, y)
    intercept, slope = lol.plot_result()

    xrange = np.linspace(0,5500,1000)
    fig, ax = plt.subplots(figsize=(7,5))
    ax.scatter(*X_whole.T)
    ax.plot(xrange, slope*xrange+intercept, c='red')
    ax.text(
        x=0.4,
        y=0.85,
        s=r'$w(x) = {a:.4f}x + {b:.4f}$'.format(a=slope, b=intercept),
        horizontalalignment='center',
        verticalalignment='center',
        transform=ax.transAxes
    )
    fig.suptitle('Weights from Gibbs')
    plt.savefig('regline_gibbs.pdf')
    plt.show()
```

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from scipy.stats import multivariate_normal, gamma, norm, rv_histogram
np.random.seed(42069)

class LinRegMetropolis:
    def __init__(self, slope=1, intercept=1, beta=1,
                 alpha=10, beta_=0.1, a=1, b=1e-2):
```

```
        self.a = a
        self.b = b
        self.alpha = alpha
        self.beta_ = beta_ # For posteriror
        self.init_slope = slope
        self.init_intercept = intercept
        self.init_beta = beta

    def fit(self, X, y, n=69000):
        self.n = n
        self.X = X
        self.y = y
        self.thetas = np.empty((n,3))
        self.thetas[0] = np.array([self.init_slope, self.init_intercept,
                                   self.init_beta])
        self.lhoods = np.empty(n)
        self.lhoods[0] = self.posterior(self.thetas[0])

        sigma = np.eye(3)*0.69
        updates = 0
        for i in range(1,n):
            proposal = multivariate_normal.rvs(mean=self.thetas[i-1], cov=
                                               sigma)
            lhood_prop = self.posterior(proposal)
            ratio = lhood_prop/self.lhoods[i-1]
            A = min((1,ratio))
            if np.random.uniform(0,1,1) < A:
                updates += 1
                self.thetas[i] = proposal
                self.lhoods[i] = lhood_prop
            else:
                self.thetas[i] = self.thetas[i-1]
                self.lhoods[i] = self.lhoods[i-1]
        return updates

    def posterior(self, theta):
        epsilon = 1e-16
        I = np.eye(2)
        origo = np.zeros(2)

        # First two elements are for linreg weights
        w = theta[:2]
        beta = theta[-1]

        logw = multivariate_normal.logpdf(w, origo, self.alpha*I).clip(
                                         epsilon)
        logbeta = gamma.logpdf(beta, self.a, self.b).clip(epsilon)
        logdata =\
            norm.logpdf(x=self.y-w@X.T, loc=0, scale=1/(beta+epsilon)).
                                               clip(epsilon).sum()

        return logw+logbeta+logdata
    def plot_result(self):
        fig, axes = plt.subplots(4,2, figsize=(7,6))
        xrange = np.arange(2,self.n+1)
        ax_left = axes[:,0].ravel()
        ax_right = axes[:,1].ravel()
        ax_left[0].plot(xrange, self.lhoods[1:], linewidth=0.69)
        ax_left[0].set_title(r'Posterior trace')
        ax_left[1].plot(xrange, self.thetas.T[0][1:], linewidth=0.69)
        ax_left[1].set_title(r'$w_1^t$ (slope) trace')
        ax_left[2].plot(xrange, self.thetas.T[1][1:], linewidth=0.69)
        ax_left[2].set_title(r'$w_0^t$ (intercept) trace')
        ax_left[3].plot(xrange, self.thetas.T[2][1:], linewidth=0.69)
        ax_left[3].set_title(r'$\beta^t$ trace')

        ax_right[0].hist(self.lhoods[1:], bins=40)
        ax_right[0].set_title(r'Posterior hist')
        ax_right[1].hist(self.thetas.T[0][1:], bins=40)
        ax_right[1].set_title(r'$w_1^t$ hist')
        ax_right[2].hist(self.thetas.T[1][1:], bins=40)
        ax_right[2].set_title(r'$w_0^t$ hist')
        ax_right[3].hist(self.thetas.T[2][1:], bins=40)
        ax_right[3].set_title(r'$\beta^t$ hist')
        fig.suptitle('Metropolis')
        fig.tight_layout()
        plt.savefig('metropolis2.pdf')
        plt.show()

        print('Enter burnout cutoff:')
        burnout = int(input())
        xrange = np.linspace(1,10,10000)
        slope_hist = rv_histogram(np.histogram(self.thetas.T[0][burnout:],
                                               bins=100))
        slope = xrange[slope_hist.pdf(xrange).argmax()]

        xrange = np.linspace(1,5000,10000)
        intercept_hist = rv_histogram(np.histogram(self.thetas.T[1][
                                              burnout:], bins=100))
        intercept = xrange[intercept_hist.pdf(xrange).argmax()]
        return intercept, slope

if __name__ == '__main__':
    df = pd.read_csv('new_york_bicycles2.csv')
    X_whole = df.values
    X = np.column_stack([X_whole[:,0], np.ones(len(X_whole))])
    y = X_whole[:,1]

    from tqdm import tqdm
    # import warnings
    # warnings.filterwarnings('ignore')

    lol = LinRegMetropolis()
    lol.fit(X,y)
    intercept, slope = lol.plot_result()
```

```
xrange = np.linspace(0,5500,1000)
fig, ax = plt.subplots(figsize=(7,5))
fig.suptitle('Weights from Metropolis')
ax.scatter(*X_whole.T)
ax.plot(xrange, slope*xrange+intercept, c='red')
ax.text(
    x=0.4,
    y=0.85,
    s=r'$w(x) = {a:.4f}x + {b:.4f}$'.format(a=slope, b=intercept),
    horizontalalignment='center',
    verticalalignment='center',
    transform=ax.transAxes
)
plt.savefig('regline_metropolis.pdf')
plt.show()
```