CSE 546 HW #3

Sam Kowash

November 29, 2018

1 Bayesian Inference

1. We consider $\{(x_i, y_i)\}_{i=1}^n$ sampled i.i.d. from a joint distribution P_{XY} over $\mathbb{R}^d \times \mathbb{R}$ satisfying $y_i \sim \mathcal{N}(x_i^T w, \sigma^2)$ for some $w \in \mathbb{R}^d$. For compactness, let $\boldsymbol{X} = [x_1, \ldots, x_n]^T$ and $\boldsymbol{y} = [y_1, \ldots, y_n]^T$.

   (a) Assume that $(\boldsymbol{X}^T\boldsymbol{X})^{-1}$ exists. The likelihood for w given a draw X, y is

$$\mathcal{L}(w \mid \boldsymbol{X}, \boldsymbol{y}) = \left(2\pi\sigma^2\right)^{-\frac{n}{2}} \exp\left[-\frac{(\boldsymbol{y} - \boldsymbol{X}w)^T(\boldsymbol{y} - \boldsymbol{X}w)}{2\sigma^2}\right],$$

   so the log likelihood is

$$\log\mathcal{L}(w \mid \boldsymbol{X}, \boldsymbol{y}) = -\frac{(\boldsymbol{y} - \boldsymbol{X}w)^T(\boldsymbol{y} - \boldsymbol{X}w)}{2\sigma^2} + \frac{n}{2}\ln(2\pi\sigma^2).$$

   Setting the gradient with respect to w to zero,

$$\begin{aligned} 0 &= -\frac{-2\boldsymbol{X}^T\boldsymbol{y} + 2\boldsymbol{X}^T\boldsymbol{X}w}{2\sigma^2} \\ \boldsymbol{X}^T\boldsymbol{X}w &= \boldsymbol{X}^T\boldsymbol{y} \\ \hat{w}_{\text{MLE}} &= (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}. \end{aligned}$$

   (b) If we assume that w is actually drawn from a Gaussian prior

$$p(w) = \frac{1}{(2\pi\tau^2)^{\frac{d}{2}}} \exp\left[-\frac{\|w\|_2^2}{2\tau^2}\right],$$

   then we can write down the posterior distribution (up to a positive constant)

$$\begin{aligned} p(w \mid \boldsymbol{X}, \boldsymbol{y}) &\propto p(\boldsymbol{X}, \boldsymbol{y} \mid w)p(w) \\ p(w \mid \boldsymbol{X}, \boldsymbol{y}) &\propto \frac{1}{\sqrt{(2\pi\sigma^2)^n(2\pi\tau^2)}} \exp\left[-\frac{(\boldsymbol{y} - \boldsymbol{X}w)^T(\boldsymbol{y} - \boldsymbol{X}w)}{2\sigma^2} - \frac{w^Tw}{2\tau^2}\right]. \end{aligned}$$

   Absorbing the prefactor into the proportionality and taking a log, we get

$$\log p(w \mid \boldsymbol{X}, \boldsymbol{y}) \propto -\frac{(\boldsymbol{y} - \boldsymbol{X}w)^T(\boldsymbol{y} - \boldsymbol{X}w)}{2\sigma^2} - \frac{w^Tw}{2\tau^2},$$

   which can be rearranged into the suggestive form

$$\log p(w \mid \boldsymbol{X}, \boldsymbol{y}) \propto -\frac{\boldsymbol{y}^T\boldsymbol{y}}{2\sigma^2} + \frac{w^T\boldsymbol{X}\boldsymbol{y} + \boldsymbol{y}^T\boldsymbol{X}^Tw}{2\sigma^2} - w^T\left(\frac{\boldsymbol{X}^T\boldsymbol{X}}{2\sigma^2} + \frac{I}{2\tau^2}\right)w.$$

Then, setting the gradient to zero,

$$\begin{aligned} 0 &= \frac{\boldsymbol{X}^T\boldsymbol{y}}{\sigma^2} - \left(\frac{\boldsymbol{X}^T\boldsymbol{X}}{\sigma^2} + \frac{I}{\tau^2}\right) w \\ \boldsymbol{X}^T\boldsymbol{y} &= \left(\boldsymbol{X}^T\boldsymbol{X} + \frac{\sigma^2}{\tau^2}\right) w \\ \hat{w}_{\text{MAP}} &= \left(\boldsymbol{X}^T\boldsymbol{X} + \frac{\sigma^2}{\tau^2}\right)^{-1} \boldsymbol{X}^T\boldsymbol{y}. \end{aligned}$$

(c) We can manipulate our previous proportionality expression

$$p(w \mid \boldsymbol{X}, \boldsymbol{y}) \propto \exp\left[-\frac{(\boldsymbol{y} - \boldsymbol{X}w)^T(\boldsymbol{y} - \boldsymbol{X}w)}{2\sigma^2} - \frac{w^Tw}{2\tau^2}\right]$$

to determine the exact form of the posterior distribution. The objective will be to complete the square into a single Gaussian in w. First, multiplying out,

$$p(w \mid \boldsymbol{X}, \boldsymbol{y}) \propto \exp\left\{-\frac{1}{2}\left[w^T\left(\frac{\boldsymbol{X}^T\boldsymbol{X}}{\sigma^2} + \frac{I}{\tau^2}\right) w - \frac{2\boldsymbol{y}^T\boldsymbol{X}w}{\sigma^2} - \frac{\boldsymbol{y}^T\boldsymbol{y}}{\sigma^2}\right]\right\}.$$

We want to find a μ such that

$$(w-\mu)^T\left(\frac{\boldsymbol{X}^T\boldsymbol{X}}{\sigma^2} + \frac{I}{\tau^2}\right)(w-\mu)$$

produces the correct cross-term in w. Expanding this expression gives,

$$\begin{aligned} (w-\mu)^T\left(\frac{\boldsymbol{X}^T\boldsymbol{X}}{\sigma^2} + \frac{I}{\tau^2}\right)(w-\mu) = w^T\left(\frac{\boldsymbol{X}^T\boldsymbol{X}}{\sigma^2} + \frac{I}{\tau^2}\right) w - 2\mu^T\left(\frac{\boldsymbol{X}^T\boldsymbol{X}}{\sigma^2} + \frac{I}{\tau^2}\right) w \\ + \mu^T\left(\frac{\boldsymbol{X}^T\boldsymbol{X}}{\sigma^2} + \frac{I}{\tau^2}\right)\mu, \end{aligned}$$

which means μ must satisfy

$$\mu^T\left(\frac{\boldsymbol{X}^T\boldsymbol{X}}{\sigma^2} + \frac{I}{\tau^2}\right) w = \frac{\boldsymbol{y}^T\boldsymbol{X}}{\sigma^2} w.$$

Since this must hold for all w, we find

$$\begin{aligned} \mu^T &= \frac{\boldsymbol{y}^T\boldsymbol{X}}{\sigma^2}\left(\frac{\boldsymbol{X}^T\boldsymbol{X}}{\sigma^2} + \frac{I}{\tau^2}\right)^{-1} \\ \mu &= \left(\boldsymbol{X}^T\boldsymbol{X} + \frac{\sigma^2}{\tau^2}\right)^{-1} \boldsymbol{X}^T\boldsymbol{y}, \end{aligned}$$

which is exactly our MAP estimator, unsurprisingly. Defining

$$\Sigma = \left(\frac{\boldsymbol{X}^T\boldsymbol{X}}{\sigma^2} + \frac{I}{\tau^2}\right)^{-1},$$

we can see that

$$p(w \mid \boldsymbol{X}, \boldsymbol{y}) \propto \exp\left\{-\frac{1}{2}\left[(w-\mu)^T\Sigma^{-1}(w-\mu) - \mu^T\Sigma^{-1}\mu - \frac{\boldsymbol{y}^T\boldsymbol{y}}{\sigma^2}\right]\right\},$$

and since the second and third terms in the exponential have no dependence on w, indeed

$$p(w \mid \boldsymbol{X}, \boldsymbol{y}) \propto \exp\left\{-\frac{1}{2}(w-\mu)^T\Sigma^{-1}(w-\mu)\right\}.$$

This is the full w-dependence of the distribution, and the constant of proportionality only enforces normalization, so we conclude that $p(w \mid \boldsymbol{X}, \boldsymbol{y}) \sim \mathcal{N}(\mu, \Sigma)$ as defined above.

(d) We now fix $z \in \mathbb{R}^d$ and define $f_z = z^T w$ as the predicted function value at z. Given the previous result, and knowing that affine transformations of normal random variables are themselves normal with μ and Σ transformed accordingly, we can see that

$$f_z \mid \boldsymbol{X}, \boldsymbol{y} \sim \mathcal{N}\left(z^T\mu, z^T\Sigma z\right),$$

or, explicitly,

$$f_z \mid \boldsymbol{X}, \boldsymbol{y} \sim \mathcal{N}\left(z^T\left(\boldsymbol{X}^T\boldsymbol{X} + \tfrac{\sigma^2}{\tau^2}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y}, z^T\left(\tfrac{\boldsymbol{X}^T\boldsymbol{X}}{\sigma^2} + \tfrac{I}{\tau^2}\right)^{-1} z\right)$$
$$f_z \mid \boldsymbol{X}, \boldsymbol{y} \sim \mathcal{N}\left(z^T\left(\boldsymbol{X}^T\boldsymbol{X} + \tfrac{\sigma^2}{\tau^2}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y}, \sigma^2 z^T\left(\boldsymbol{X}^T\boldsymbol{X} + \tfrac{\sigma^2}{\tau^2}\right)^{-1} z\right).$$

(e) It is an identity that for matrices A, U, C, V with A nonsingular, we have

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U\left(C^{-1} + VA^{-1}U\right)^{-1}VA^{-1}.$$

Taking

$$\begin{aligned} A &= \frac{\sigma^2}{\tau^2}I \\ U &= \boldsymbol{X}^T \\ C &= I \\ V &= \boldsymbol{X}, \end{aligned}$$

we get

$$\begin{aligned} \left(\frac{\sigma^2}{\tau^2}I + \boldsymbol{X}^T\boldsymbol{X}\right)^{-1} &= \frac{\tau^2}{\sigma^2}I - \frac{\tau^2}{\sigma^2}\boldsymbol{X}^T\left(I + \boldsymbol{X}\frac{\tau^2}{\sigma^2}\boldsymbol{X}^T\right)^{-1}\boldsymbol{X}\frac{\tau^2}{\sigma^2} \\ \left(\frac{\sigma^2}{\tau^2}I + \boldsymbol{X}^T\boldsymbol{X}\right)^{-1} &= \frac{\tau^2}{\sigma^2}I - \frac{\tau^2}{\sigma^2}\boldsymbol{X}^T\left(\frac{\sigma^2}{\tau^2}I + \boldsymbol{X}\boldsymbol{X}^T\right)^{-1}\boldsymbol{X}. \end{aligned}$$

Now defining

$$\begin{aligned} \boldsymbol{K} &= \boldsymbol{X}\boldsymbol{X}^T \\ \boldsymbol{k}_{\cdot z} &= \boldsymbol{X}z \\ k_{zz} &= z^T z, \end{aligned}$$

we can rewrite $z^T\mu$ as

$$z^T\mu = z^T\left(\boldsymbol{X}^T\boldsymbol{X} + \frac{\sigma^2}{\tau^2}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y} = \frac{\tau^2}{\sigma^2}z^T\left[I - \boldsymbol{X}^T\left(\frac{\sigma^2}{\tau^2}I + \boldsymbol{X}\boldsymbol{X}^T\right)^{-1}\boldsymbol{X}\right]\boldsymbol{X}^T\boldsymbol{y}$$

$$z^T\mu = \frac{\tau^2}{\sigma^2}\left[\boldsymbol{k}_{\cdot z}^T\boldsymbol{y} - \boldsymbol{k}_{\cdot z}^T\left(\frac{\sigma^2}{\tau^2}I + \boldsymbol{K}\right)^{-1}\boldsymbol{K}\boldsymbol{y}\right].$$

If we insert an identity in the middle of the first term as $(\frac{\sigma^2}{\tau^2}I + \boldsymbol{K})^{-1}(\frac{\sigma^2}{\tau^2}I + \boldsymbol{K})$, then

$$z^T\mu = \frac{\tau^2}{\sigma^2}\boldsymbol{k}_{\cdot z}^T\left(\frac{\sigma^2}{\tau^2}I + \boldsymbol{K}\right)^{-1}\left[\frac{\sigma^2}{\tau^2}I + \boldsymbol{K} - \boldsymbol{K}\right]\boldsymbol{y}$$
$$z^T\mu = \boldsymbol{k}_{\cdot z}^T\left(\frac{\sigma^2}{\tau^2}I + \boldsymbol{K}\right)^{-1}\boldsymbol{y}.$$

Similarly, we can rewrite the variance (in fewer steps) as

$$\sigma^2 z^T\Sigma z = \sigma^2 z^T\left(\boldsymbol{X}^T\boldsymbol{X} + \tfrac{\sigma^2}{\tau^2}\right)^{-1} z = \tau^2\left[k_{zz} - \boldsymbol{k}_{\cdot z}^T\left(\tfrac{\sigma^2}{\tau^2}I + \boldsymbol{K}\right)^{-1}\boldsymbol{k}_{\cdot z}\right].$$

We then see that our MAP estimate (the mean of this distribution) for f_z is exactly the prediction made by kernel ridge regression with a linear kernel given by K and regularization σ^2/τ^2.

2. Let $\{(x_i, y_i)\}_{i=1}^n$ be drawn i.i.d. from a distribution P_{XY} over $\mathbb{R}^d \times \mathbb{R}$ such that for some $w \in \mathbb{R}^d$ we have $y_i \sim \mathcal{N}(x_i^T w, \sigma^2)$.

   (a) Suppose that w is drawn from a Laplace prior

$$p(w) = \frac{1}{(2a)^d}\exp\left(-\frac{\|w\|_1}{a}\right).$$

   The posterior distribution then satisfies

$$p(w \mid \boldsymbol{X}, \boldsymbol{y}) \propto \exp\left[-\frac{(\boldsymbol{y} - \boldsymbol{X}w)^T(\boldsymbol{y} - \boldsymbol{X}w)}{2\sigma^2} - \frac{\|w\|_1}{a}\right].$$

   Maximizing this probability is minimizing

$$\frac{(\boldsymbol{y} - \boldsymbol{X}w)^T(\boldsymbol{y} - \boldsymbol{X}w)}{2\sigma^2} + \frac{\|w\|_1}{a},$$

   which is LASSO regression with $\lambda = 1/a$. We know that this does not have a general closed form for $d > 1$ so can say simply

$$\hat{w}_{\text{MAP}} = \hat{w}_{\text{LASSO}},$$

   and use our existing coordinate descent algorithm based on the subgradient of the LASSO objective to find particular solutions.

   (b)

2 Kernel Regression

3. (a) LOOCV was temperamental in this problem; when I tried alternately minimizing λ and the hyperparameter for a given kernel, it was generally driven toward the extremes of the parameter range, e.g. $\lambda = 10^{\pm 8}$ and $d = 0$ or 60, for one run. This made it difficult to pick these parameters in a principled way.

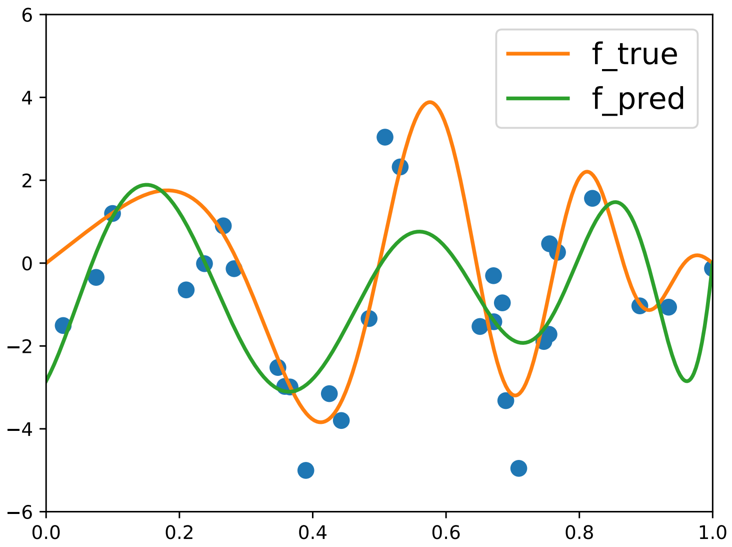

Figure 2.1: Predicted and actual functions with polynomial kernel, $d = 20$, $\lambda = 10^{-4}$

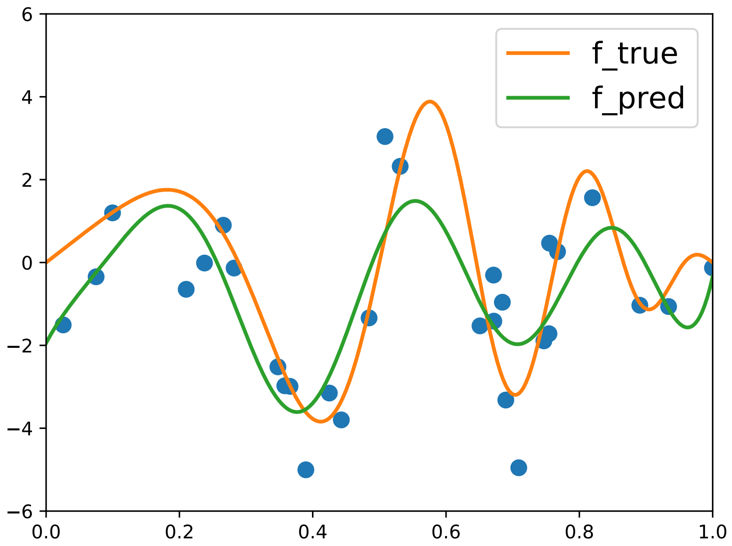

Figure 2.2: Predicted and actual functions with rbf kernel, $\gamma = 10$, $\lambda = 10^{-4}$

(b)

(c)

(d)

(e)

3 k-means clustering

4. (a) Figures below show cluster centers in the full MNIST set (train joined with test) from k-means for a variety of different cluster counts. In general, they all resemble identifiable numerals, but more clusters tends to lead to more distinct features. At $k = 5$ we see centers somewhere in between similar numerals like 3 and 8 or 9 and 4, but at $k = 20$ the algorithm has the freedom to center clusters around more specific forms like taller or rounder or more slanted 9s.

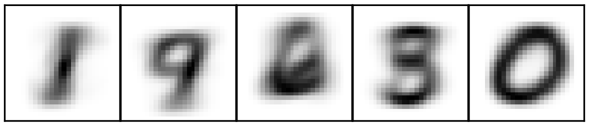

Figure 3.1: Final centers for $k = 5$ with uniform initialization

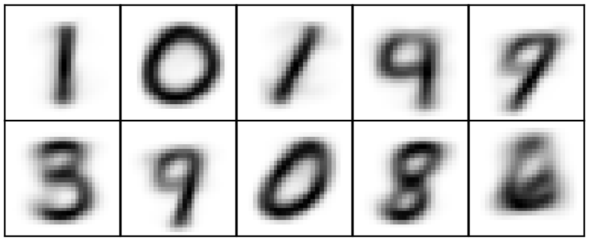

Figure 3.2: Final centers for $k = 10$ with uniform initialization

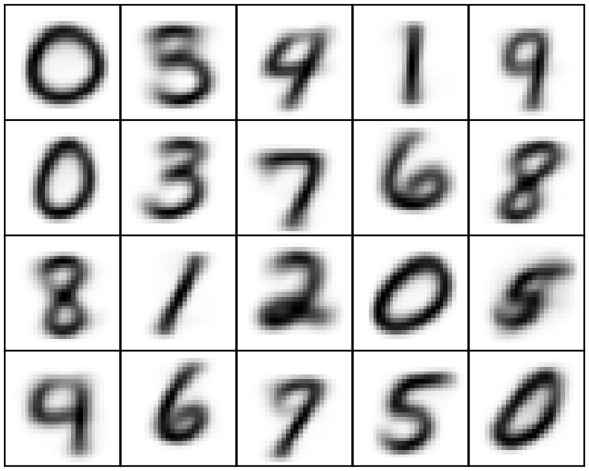

Figure 3.3: Final centers for $k = 20$ with uniform initialization

(b) Figures below show centers obtained with the `k-means++` intialization scheme. They look broadly similar to the uniform case, although perhaps modestly crisper.

Fig. 4b compare the convergence rate to uniform initialization through the mean squared distance from a point to its cluster center. We see that other than for $k = 5$, the new initialization converges faster and to smaller mean distances.

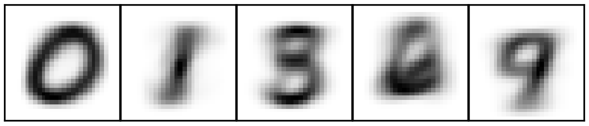

Figure 3.4: Final centers for $k = 5$ with `k-means++` initialization

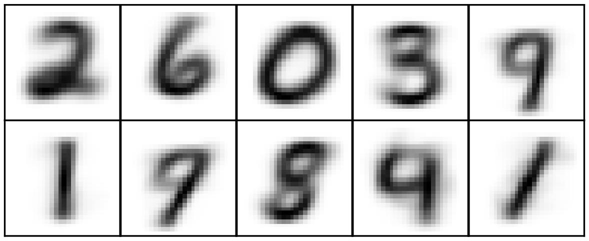

Figure 3.5: Final centers for $k = 10$ with `k-means++` initialization

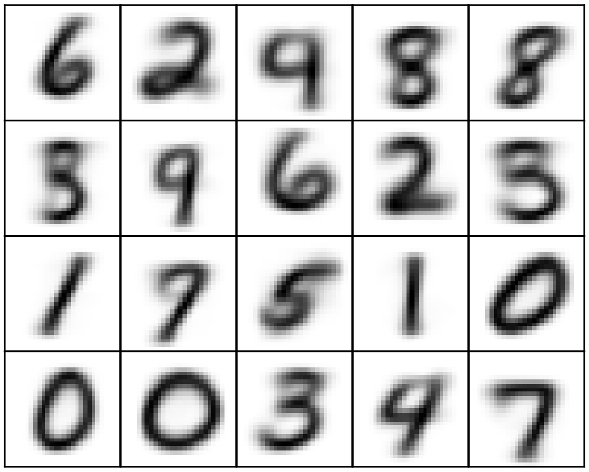

Figure 3.6: Final centers for $k = 20$ with `k-means++` initialization

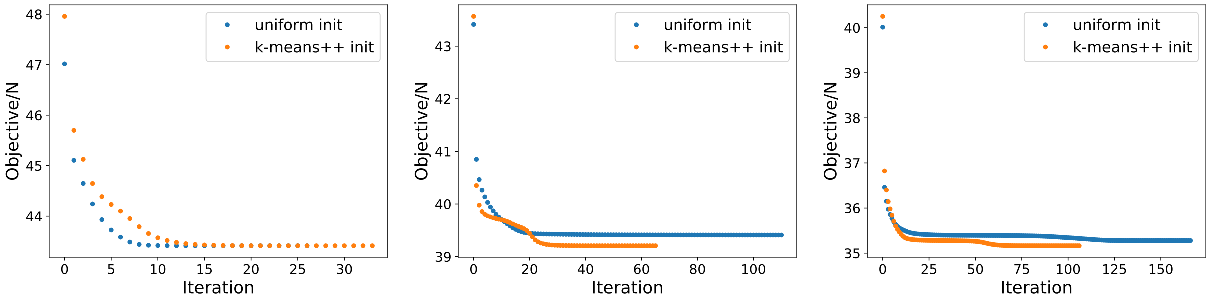

Figure 3.7: Objective function vs. iteration number for $k = 5, 10, 20$ from left to right

4 Joke Recommender System

5. (a)

   (b)

   (c)

```
#!/usr/bin/env python
import numpy as np
import matplotlib.pyplot as plt
import sys

class Kernel:
    def __init__(self, X, ker_name=’poly’, hyper=None):
        self.X = X
        self.set_ker(ker_name,hyper)


    def set_ker(self,ker_name,hyper=None):
        if ker_name == ’poly’:
            self.name = ’poly’
            n = len(self.X)
            self.base_func = lambda x1,x2: 1 + x1*x2
            self.base_mat = np.ones((n,n)) + np.outer(self.X,self.X) #1-d only
            self.hyper = hyper if hyper is not None else 0

        elif ker_name == ’rbf’:
            self.name = ’rbf’
            n = len(self.X)
            XX = np.repeat([self.X],n,axis=0)
            Xdiff = XX - XX.T
            self.base_func = lambda x1,x2: np.exp(-1*np.dot(x1-x2,np.transpose(x1-x2)))
            self.base_mat = np.exp(-1*np.power(Xdiff,2)) #works for 1-d data
            self.hyper = (hyper if hyper is not None
                          else 1/np.median(np.abs(Xdiff[np.triu_indices(m,1)])**2))

        else:
            print("Bad␣kernel␣name.")
            sys.exit()

    def get_matrix(self):
        return np.power(self.base_mat,self.hyper)

    def get_func(self):
        return lambda x1,x2: np.power(self.base_func(x1,x2),self.hyper)

    def get_predictor(self,w):
        func = self.get_func()
        return lambda x: np.dot(w, np.array([func(xi,x) for xi in self.X]))



def train_ker_ridge(ker_mat,Y_train,lam):
    n = ker_mat.shape[0]
    return np.linalg.solve(ker_mat+lam*np.identity(n),Y_train)



def loocv_lam(X, Y, hyper, ker_name=’poly’):
    lams = 10.**np.arange(-7,-1)
    val_errs = np.zeros_like(lams)

    for i in range(len(lams)):
        print(f"On␣lambda={lams[i]:2f}")
        x_inds = np.arange(len(X))
        errs = np.zeros_like(x_inds)

        for ind in x_inds:
            X_train = X[x_inds != ind]
            Y_train = Y[x_inds != ind]

            ker = Kernel(X_train,ker_name,hyper)
            ker_mat = ker.get_matrix()

            X_val = X[ind]
            Y_val = Y[ind]

            w = train_ker_ridge(ker_mat, Y_train, lams[i])
            predictor = ker.get_predictor(w)
            errs[ind] = np.abs(predictor(X_val)-Y_val)**2

        val_errs[i] = np.mean(errs)

    return lams, val_errs


def loocv_hyper(X, Y, hypers, lam, ker_name=’poly’):
    val_errs = np.zeros_like(hypers)

    for i in range(len(hypers)):
        print(f"On␣hyper={hypers[i]}")
        x_inds = np.arange(len(X))

        errs = np.zeros_like(x_inds)

        for ind in x_inds:
            X_train = X[x_inds != ind]
            Y_train = Y[x_inds != ind]

            ker = Kernel(X_train,ker_name,hypers[i])
            ker_mat = ker.get_matrix()

            X_val = X[ind]
            Y_val = Y[ind]

            w = train_ker_ridge(ker_mat, Y_train, lam)
            predictor = ker.get_predictor(w)
            errs[ind] = np.abs(predictor(X_val)-Y_val)**2

        val_errs[i] = np.mean(errs)

    return val_errs


f = lambda x: 4*np.sin(np.pi*x)*np.cos(6*np.pi*np.power(x,2))
n = 30
X = np.random.uniform(0,1,n)
Y = f(X) + np.random.randn(n)

ker = Kernel(X,’poly’,20)
ker_mat = ker.get_matrix()

w = train_ker_ridge(ker_mat,Y, 1e-4)
predictor = ker.get_predictor(w)
```

```
x = np.linspace(0,1,1000)
y = np.array([predictor(xx) for xx in x])

plt.plot(X,Y,’o’,ms=8)
plt.plot(x,f(x),’-’,lw=2,label=’f_true’)
plt.plot(x,y,’-’,lw=2,label=’f_pred’)
plt.xlim((0,1))
plt.ylim((-6,6))
plt.legend(fontsize=16)
plt.savefig(’../figures/poly_ker_d20_lm4.pdf’,bbox_inches=’tight’)
plt.cla()




ker = Kernel(X,’rbf’,10)
ker_mat = ker.get_matrix()

w=train_ker_ridge(ker_mat,Y,1e-4)
predictor = ker.get_predictor(w)

x = np.linspace(0,1,1000)
y = np.array([predictor(xx) for xx in x])

plt.plot(X,Y,’o’,ms=8)
plt.plot(x,f(x),’-’,lw=2,label=’f_true’)
plt.plot(x,y,’-’,lw=2,label=’f_pred’)
plt.xlim((0,1))
plt.ylim((-6,6))
plt.legend(fontsize=16)
plt.savefig(’../figures/rbf_ker_gp5_lm4.pdf’,bbox_inches=’tight’)



# lams,val_errs = loocv_lam(X,Y,20,’poly’)
# plt.plot(lams,val_errs,’o-’)
# plt.xscale(’log’)
# plt.show()
#
#
# val_errs = loocv_hyper(X,Y,np.arange(0,30,2),1e-4,’poly’)
# plt.plot(np.arange(0,30,2),val_errs,’o-’)
# plt.show()
```

```python
#!/usr/bin/env python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec


# k-means centers
dim_dict = {5 : (1,5), 10 : (2,5), 20 : (4,5)}


for k in [5,10,20]:
    for pp in [True,False]:
        fig = plt.figure(figsize=tuple(reversed(dim_dict[k])))
        gs = fig.add_gridspec(*dim_dict[k])
        gs.update(wspace=0.,hspace=0.)

        pp_str = 'pp' if pp else ''

        data = np.load(f'data/{k}-clusters{pp_str}.npz')
        objs, mu = data['objs']/70000,data['mu']

        for i in range(k):
            ind = np.unravel_index(i, dim_dict[k])
            ax = fig.add_subplot(gs[ind])
            ax.set_xticks([])
            ax.set_yticks([])
            ax.set_xticklabels([])
            ax.set_yticklabels([])
            ax.imshow(mu[i].reshape((28,28)),aspect='equal',cmap='binary')

    plt.savefig(f'../figures/{k}{pp_str}-centers.pdf',bbox_inches='tight')
    plt.clf()

    data = np.load(f'data/{k}-clusters.npz')
    datapp = np.load(f'data/{k}-clusterspp.npz')

    objs,objspp = data['objs']/70000, datapp['objs']/70000

    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.plot(objs,'o',ms=4,label="uniform␣init")
    ax.plot(objspp,'o',ms=4,label="k-means++␣init")
    ax.set_xlabel('Iteration', size=18)
    ax.set_ylabel('Objective/N', size=18)
    ax.tick_params(labelsize=14)
    ax.legend(fontsize=16)
    plt.savefig(f'../figures/{k}-objective.pdf',bbox_inches='tight')
    plt.clf()
```

```
#!/usr/bin/env python
import numpy as np
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
from functools import partial
from mnist import MNIST
from multiprocessing import Pool
import os
import sys



def load_test():
    mndata = MNIST('../../mnist/data/')
    X_test, labels_test = map(np.array, mndata.load_testing())
    X_test = X_test/255.

    return X_test, labels_test

def load_train():
    mndata = MNIST('../../mnist/data/')
    X_train, labels_train = map(np.array, mndata.load_training())
    X_train = X_train/255.

    return X_train, labels_train



def kpp_init(k, X):
    n,d = X.shape

    mu = [X[np.random.randint(n)]] #choose first cluster uniformly
    print("Center␣1␣placed.")

    assignments = assign(X,mu)

    while len(mu) < k:
        prob = np.zeros(n)
        for assn in assignments:
            for j in assignments[assn]:
                prob[j] = np.dot(X[j]-mu[assn],X[j]-mu[assn])

        prob /= np.sum(prob)
        mu.append(X[np.random.choice(n,p=prob)])
        print(f"Center␣{len(mu)}␣placed.")
        assignments = assign(X,mu)

    return np.array(mu),assignments

def get_cluster(mu,X):
    k = len(mu)
    return np.argmin([np.dot(X-mu[i],X-mu[i]) for i in range(k)])

def assign(X,mu):
    n,d = X.shape
    k = len(mu)
    assignments = {i: [] for i in range(k)}
```

```
#     assign_inds = np.fromiter(map(partial(get_cluster,mu),X),
#                                dtype=int,count=n)
    with Pool(7) as pool:
        assign_inds = np.fromiter(pool.map(partial(get_cluster,mu), X),
                                  dtype=int,count=n)

    for i in range(k): assignments[i] = np.arange(n)[np.where(assign_inds == i)]

    return assignments

def recenter(X,assignments):
    k = len(assignments)
    n,d = X.shape

    mu = [np.mean(X[assignments[i],:],axis=0) for i in assignments]

    return mu

def objective(X,mu,assignments):
    k = len(mu)
    err = np.zeros(k)

    for i in range(k):
        dev = X[assignments[i],:] - mu[i]
        dev *= dev
        err[i] = np.sum(dev)

    return np.sum(err)

if __name__ == '__main__':
    X_train, labels_train = load_train()
    X_test, labels_test = load_test()

    X = np.vstack((X_train,X_test))
    n,d = X.shape
    print("Data␣loaded.")

    k = int(sys.argv[1])
    thresh = 1e-3

    if sys.argv[2] == 'uni':
        seeds = np.random.permutation(n)[:k]
        mu = X[seeds]
        mu_last = np.copy(mu)

        assignments = assign(X,mu)
        assignments_last = assignments.copy()
        print("Clusters␣seeded.")
    elif sys.argv[2] == 'kpp':
        mu, assignments = kpp_init(k,X)
        mu_last = np.copy(mu)
        assignments_last = assignments.copy()
    else: print("Bad␣arg,␣specify␣uni␣or␣kpp")

    mu = recenter(X,assignments)
    print("First␣recentering")

    objs = [objective(X,mu,assignments)]
    i = 0
    assignments = assign(X,mu)
    mu = recenter(X,assignments)
    print("Manual␣first␣step.")


    print("Entering␣loop.")
    while np.any([np.any(assignments_last[i] != assignments[i]) for i in assignments]):
        i += 1
        print(f"On␣iteration␣{i:4d},␣last␣delta:␣{np.max(np.abs(mu-mu_last)):4f}")

        mu_last = np.copy(mu)
        assignments_last = assignments.copy()

        assignments = assign(X,mu)
        mu = recenter(X,assignments)

        objs.append(objective(X,mu,assignments))

    print("Escaped␣the␣loop.")

    fname = f'{k}-clusters' if sys.argv[2] == 'uni' else f'{k}-clusterspp'


    np.savez(f'data/{fname}',objs=np.array(objs),mu=mu)

#     for i in range(k):
#         plt.imshow(mu[i].reshape(28,28),aspect='equal',cmap=plt.get_cmap('binary'))
#         plt.tight_layout()
#         plt.savefig(f'../figures/{fname}-{i}.pdf')
#
#     plt.cla()
#     plt.clf()
#
#     plt.plot(np.log10(objs),'o',ms=3)
#     plt.xlabel('Iteration', size=18)
#     plt.ylabel('log(objective)', size=18)
#
#     plt.tight_layout()
#     plt.savefig(f'../figures/{fname}_objective.pdf')
```

```
#!/usr/bin/env python
import numpy as np
import scipy.sparse as sp
from scipy.sparse.linalg import svds


def parse_raw(raw):
    n_users = int(raw[-1,0])
    n_jokes = 100

    users = raw[:,0].astype(int)-1
    jokes = raw[:,1].astype(int)-1

    result = sp.coo_matrix((raw[:,2], (users, jokes))).tocsc()
    means = np.array(result.sum(axis=0)/result.getnnz(axis=0)).flatten()
    demeaned = sp.coo_matrix((raw[:,2]-means[jokes], (users,jokes)))


    return result, demeaned.tocsc(), means


def mse(R_pred,test_raw):
    n_data = len(test_raw)
    users = test_raw[:,0].astype(int)-1
    jokes = test_raw[:,1].astype(int)-1
    errs = R_pred[users,jokes] - test_raw[:,2]

    return np.sum(errs**2)/n_data

def mae(R_pred,test,test_raw):
    n_users,n_jokes = test.shape
    n_ratings = test.getnnz(axis=1)

    users = test_raw[:,0].astype(int)-1
    jokes = test_raw[:,1].astype(int)-1

    err = 0
    for i in range(n_users):
        ind = np.where(users == i)
        err_i = R_pred[users[ind],jokes[ind]] - test_raw[ind,2]
        err += np.sum(np.abs(err_i))/n_ratings[i]

    return err/n_users


def train_dumb(train):
    n_users,n_jokes = train.shape
    return (np.ones((n_users,1)),
            np.array(train.sum(axis=0)/train.getnnz(axis=0)))


def train_svd(train,d):
    U,S,Vt = svds(train,k=d)
    return U, sp.diags(S,0).dot(Vt)

def pred(U,Vt,means):
    n = U.shape[0]
    return U.dot(Vt) + np.outer(np.ones(n), means)
```

```
train_raw = np.genfromtxt('data/jester/train.txt',delimiter=',')
test_raw = np.genfromtxt('data/jester/test.txt',delimiter=',')




train, d_train, means_train = parse_raw(train_raw)
test, d_test, means_test = parse_raw(test_raw)

# U,Vt = train_dumb(d_train)
# R_pred = pred(U,Vt,means_train)

ds = [1,2,5,10,20,50]
mses_train = []
mses_test = []
maes_train = []
maes_test = []


for d in ds:
    print(f"On␣d={d}")
    U,Vt = train_svd(d_train,d)
    R_pred = pred(U,Vt,means_train)

    mses_test.append(mse(R_pred,test_raw))
    maes_test.append(mae(R_pred,test,test_raw))
    mses_train.append(mse(R_pred,train_raw))
    maes_train.append(mae(R_pred,train,train_raw))

ds = np.array(ds)
mses_train = np.array(mses_train)
maes_train = np.array(maes_train)
mses_test = np.array(mses_test)
maes_test = np.array(maes_test)
np.savez('data/svd.npz',d=ds,mse_train=mses_train,mae_train=maes_train,
         mse_test=mses_test,mae_test=maes_test)
```