STAT 37710: Machine Learning	Autumn 2022

Homework 2 Solutions

Please do not distribute.

1. MAP (*20 points*)

Assume that for each $1 \le i \le n$, one has $y_i = w^\top x_i + \varepsilon_i$ with $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$.

a.(*10 points*) Assume a prior $w \sim \mathcal{N}(0, \tau^2 I_d)$, where I_d is the identity matrix in $d \times d$ dimension. Show that the MAP estimator in this case is equivalent to the ridge regression estimator. Please be precise about the choice of the regularization parameter λ.

b.(*10 points*) Assume a prior $p(w) = (\tau/2)^d \exp(-\tau \|w\|_1)$, where τ is a parameter for this prior. Show that the MAP estimator in this case is equivalent to the Lasso estimator. Please be precise about the choice of the regularization parameter λ.

Solution:

a. The joint pdf (and hence the conditional pdf of w) as a function of w is proportional to

$$\exp\left(-\frac{1}{2}\frac{\|w\|_2^2}{\tau^2} - \sum_{i=1}^{n} \frac{(y_i - w^T x_i)^2}{2\sigma^2}\right)$$

Maximizing this over w is equivalent to minimizing

$$\sum_{i=1}^{n} (y_i - w^T x_i)^2 + \frac{\sigma^2}{\tau^2}\|w\|_2^2$$

so $\lambda = \frac{\sigma^2}{\tau^2}$

b. The joint pdf is proportional to

$$\exp\left(-\tau\|w\|_1 - \sum_{i=1}^{n} \frac{(y_i - w^T x_i)^2}{2\sigma^2}\right)$$

$\lambda = 2\sigma^2\tau.$

2. Regression function and Bayes classifier (*20 points*)

Consider a binary classification problem with $\mathcal{Y} = \{0, 1\}$. Let $g^\star(x)$ be the Bayes optimal classifier, and $m^\star(x)$ be the optimal regression function. Let $\hat{m}$ be a fixed function from $\mathcal{X}$ to $\mathbb{R}$. Define the plug-in decision $\hat{g}$ by

$$\hat{g}(x) = \begin{cases} 1, & \text{if } \hat{m}(x) \ge 1/2, \\ 0, & \text{if } \hat{m}(x) < 1/2. \end{cases}$$

Prove the following statements.

$$\begin{aligned} 0 &\le \mathbb{P}(\hat{g}(X) \ne Y) - \mathbb{P}(g^\star(X) \ne Y) \\ &\le 2\mathbb{E}_X[|\hat{m}(X) - m^\star(X)|] \le 2(\mathbb{E}_X[|\hat{m}(X) - m^\star(X)|^2])^{1/2}. \end{aligned}$$

Solution: *Note the definition of $m^*(x) := P(Y = 1|x)$. The first inequality follows from the definition of $g^*(\cdot)$, which evaluates to 1 if $m^*(x) \geq 1/2$ and 0 otherwise. The third inequality is an application of Jensen's inequality. As for the second inequality,*

$$
\begin{aligned}
P(\hat{g}(X) \neq Y) - P(g^*(X) \neq Y) &= \int\int 1_{\hat{g}(x)\neq y} - 1_{g^*(x)\neq y} dP(y|x)dP(x) \\
&= \int\int 1_{y\neq g^*(x)} 1_{\hat{g}(x)=g^*(x)} + 1_{y=g^*(x)} 1_{\hat{g}(x)\neq g^*(x)} - 1_{y\neq g^*(x)} dP(y|x)dP(x) \\
&= \int 1_{\hat{g}(x)\neq g^*(x)} \left(\int 1_{y=g^*(x)} - 1_{y\neq g^*(x)} dP(y|x)\right) dP(x) \\
&= \int 1_{\hat{g}(x)\neq g^*(x)} |1 - 2m^*(x)| dP(x)
\end{aligned}
$$

where the last equality follows from $\int 1_{y=g^*(x)} dP(y|x) = \max(m^*(x), 1 - m^*(x))$ (by definition of $g^*(\cdot)$ and $m^*(\cdot)$), which implies

$$\int 1_{y=g^*(x)} - 1_{y\neq g^*(x)} dP(y|x) = |1 - 2m^*(x)|$$

We further have

$$
\begin{aligned}
&\int 1_{\hat{g}(x)\neq g^*(x)} |1 - 2m^*(x)| dP(x) \\
&= \int_{\hat{m}(x)\geq 1/2, m^*(x)<1/2} 1 - 2m^*(x) dP(x) + \int_{\hat{m}(x)<1/2, m^*(x)\geq 1/2} 2m^*(x) - 1 dP(x) \\
&\leq \int_{\hat{m}(x)\geq 1/2, m^*(x)<1/2} 2\hat{m}(x) - 2m^*(x) dP(x) + \int_{\hat{m}(x)<1/2, m^*(x)\geq 1/2} 2m^*(x) - 2\hat{m}(x) dP(x) \\
&\leq 2 \int |\hat{m}(x) - m^*(x)| dP(x)
\end{aligned}
$$

3. Multi-class logistic regression (*20 points***)** The posterior probabilities for mulitclass logistic regression can be given as a softmax transformation of hyperplanes, such that:

$$P(Y = k \mid \mathbf{X} = \mathbf{x}) = \frac{\exp(\mathbf{a}_k^\top \mathbf{x})}{\sum_j \exp(\mathbf{a}_j^\top \mathbf{x})}$$

If we consider the use of maximum likelihood to determine the parameters $\mathbf{a}_k$, we can take the negative logarithm of the likelihood function to obtain the *cross-entropy* error function for multiclass logistic regression:

$$E(\mathbf{a}_1, \ldots, \mathbf{a}_K) = -\ln\left(\prod_{n=1}^{N}\prod_{k=1}^{K} P(Y = k \mid \mathbf{X} = \mathbf{x}_n)^{t_{nk}}\right) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \ln y_{nk}$$

where $t_{nk} = 1\{\text{labelOf}(\mathbf{x}_n) = k\}$, and $y_{nk} = P(Y = k \mid \mathbf{X} = \mathbf{x}_n)$.

a.**(***10 points***)** For $j \in \{1, \ldots, K\}$, prove that

$$\frac{\partial t_{nk} \ln y_{nk}}{\partial \mathbf{a}_j} = t_{nk}(1\{k = j\} - y_{nj})\mathbf{x}_n$$

b.(*10 points*) Based on the result in (a), show that the gradient of the error function can be stated as given below

$$\nabla_{\mathbf{a}_k} E(\mathbf{a}_1, \ldots, \mathbf{a}_K) = \sum_{n=1}^{N} [y_{nk} - t_{nk}]\, \mathbf{x}_n$$

Solution:

a. Note that

$$\frac{d}{d\boldsymbol{a}_j} \log y_{nk} = x_n 1_{k=j} - \frac{x_n \exp(a_j^T x_n)}{\sum_l \exp(a_l^T x_n)} = x_n(1_{k=j} - y_{nj})$$

b. From the previous part,

$$\frac{dE}{da_j} = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}(1_{k=j} - y_{nj})x_n = -\sum_{n=1}^{N} x_n(t_{nj} - y_{nj}\sum_k t_{nk}) = \sum_n x_n(y_{nj} - t_{nj})$$

4. Programming assignment: Lasso (*40 points*)

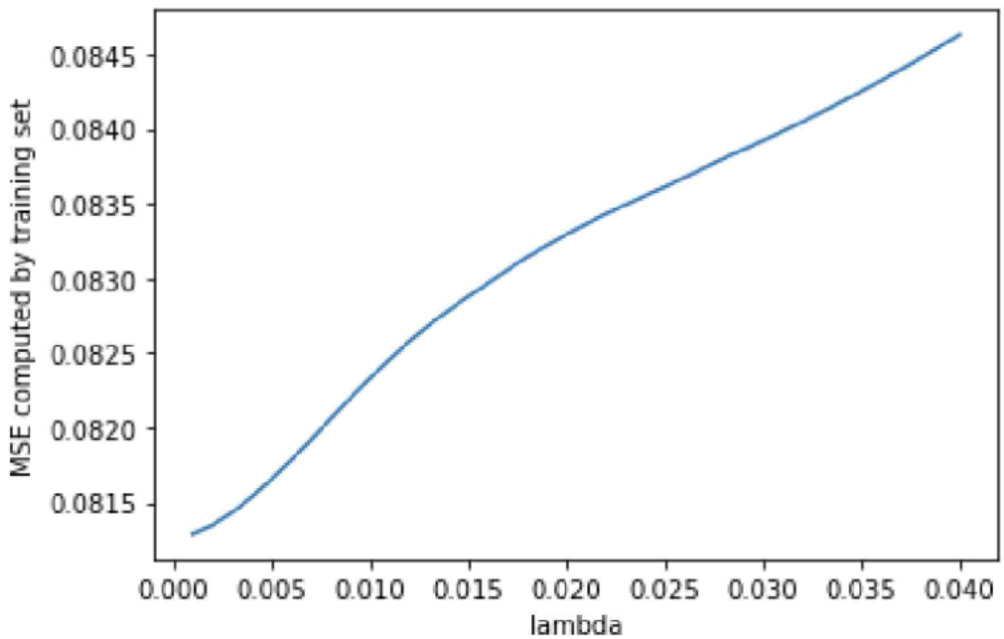
In this question, you are required to fit data with a Lasso regression. Recall that the Lasso objective is to minimize $\text{RSS}(\beta) + \lambda \sum_i \beta_i$. Following the script below here, you will be able to generate the training and testing data.

```
#python
import numpy as np
np.random.seed(0)
N_fold = 10
N_test = 500
N_train = 1000
N = N_test + N_train
# Specify feature dimensions of X and Y
X_dim = 20
Y_dim = 10
X = np.random.randn(N,X_dim)

# Only have 10 non-zero entries in beta,
nnz = 10
beta = np.zeros((X_dim * Y_dim))
nnz_idx = np.random.choice(X_dim * Y_dim, nnz, replace = False)
beta[nnz_idx] = np.random.randn(nnz) * 2

beta = beta.reshape(X_dim, Y_dim)
Y =  X @ beta + np.random.rand(N, Y_dim)

# Split training and testing set
X_test = X[:N_test]
Y_test = Y[:N_test]
X_train = X[N_test:]
Y_train = Y[N_test:]
```

a.(*20 points*) Write a function to fit the Lasso regression on the training data and calculate the MSE on the training set. Choosing λ from 0 to 0.04 (with a step of 0.001), compute the estimate $\hat{y}$ for different values λ, and plot the MSE as a function of λ.

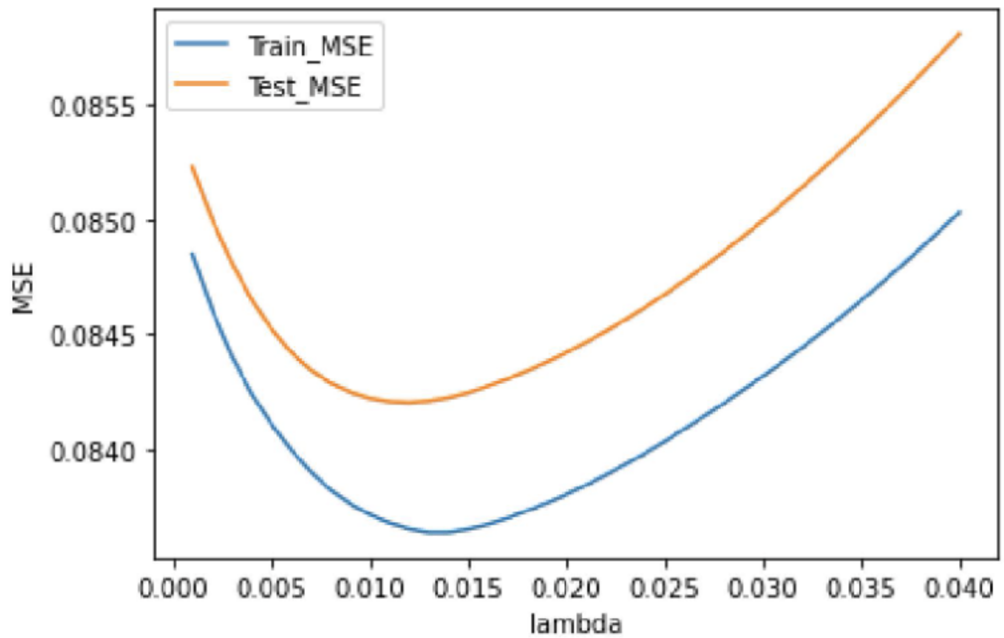
b.(*20 points*) Implement 10-fold cross validation on the training set to select λ. Plot and compare the MSE on the hold-out set with the true MSE which is computed on the test set. And see how we get to finding the "best" λ.

Solution:



a.

```
plt.show()
```




```
[6]: i = np.argmin(Test_MSE)
print("The minimum test MSE is {}, when lambda is {}. And this is the best␣
 ↪lambda.".format(Test_MSE[i],Lam[i]))
```

```
The minimum test MSE is 0.08420399983542662, when lambda is 0.012. And this is
the best lambda.
```

b.

```
#python
def train_and_eval(X_train, Y_train, X_eval, Y_eval,
lambda_):
    clf = linear_model.Lasso(alpha=lambda_)
    clf.fit(X_train, Y_train)
    Y_eval_pred = clf.predict(X_eval)
    mse = ((Y_eval - Y_eval_pred) ** 2).sum(axis = -1).mean(
    axis = 0)
    return mse

weight_list = np.arange(40)[1:] * 0.001
result_list = []
```

```
for weight in weight_list:
    test_mse = train_and_eval(X_train,
    Y_train, X_train, Y_train, weight)
    result_list.append([test_mse, weight])
result_array = np.array(result_list)
plt.figure()
plt.plot(result_array[:,-1], result_array[:,0], label =
'train mse')
plt.xlabel('lambda')
plt.ylabel('train mse')
plt.legend()
plt.show()
```

```
#python
weight_list = np.arange(40)[1:] * 0.001
result_list = []
for weight in weight_list:
    cv_eval_mse_list = []
    for i in range(N_fold):
        sidx = i * int(N_train / N_fold)
        eidx = (i+1)* int(N_train / N_fold)
        x_cv_eval = X_train[sidx:eidx]
        y_cv_eval = Y_train[sidx:eidx]
        x_cv_train = np.concatenate([X_train[:sidx],
        X_train[eidx:]],axis = 0)
        y_cv_train = np.concatenate([Y_train[:sidx],
        Y_train[eidx:]],axis = 0)
        eval_mse = train_and_eval(x_cv_train, y_cv_train,
        x_cv_eval, y_cv_eval, weight)
        cv_eval_mse_list.append(eval_mse)
    test_mse = train_and_eval(X_train, Y_train, X_test,
    Y_test, weight)
    result_list.append([np.array(cv_eval_mse_list).mean(),
    test_mse, weight])
result_array = np.array(result_list)
plt.figure()
plt.plot(result_array[:,-1], result_array[:,1], label =
'gt mse')
plt.plot(result_array[:,-1], result_array[:,0], label =
'cv mse')
plt.xlabel('lambda')
plt.ylabel('mse')
plt.legend()
plt.show()
```