Homework #3

CSE 546: Machine Learning
Prof. Kevin Jamieson
Due: 11/21 11:59 PM

1 Analysis of Variance

1. *[7 points]* For $i = 1, \dots, n$ fix $x_i \in \mathbb{R}^d$ and let $y_i = x_i^T w_* + \epsilon_i$ where $\epsilon_i \overset{i.i.d.}{\sim} \mathcal{N}(0, \sigma^2)$. Let $\widehat{w} = \arg\min_{w \in \mathbb{R}^d} \sum_{i=1}^n (y_i - x_i^T w)^2$. In the last homework you showed that $\mathbb{E}_\epsilon[||\widehat{w} - w_*||_2^2] = \sigma^2 Tr((\mathbf{X}^T\mathbf{X})^{-1})$. In this problem, we will study in what directions $\widehat{w}$ is most accurate.
Let $\mathbf{V}\Lambda\mathbf{V}^T$ be the eigenvalue decomposition of $\mathbf{X}^T\mathbf{X}$ such that $\mathbf{V}^T\mathbf{V} = I$ and Λ is a matrix of all zeros with $(\lambda_1, \dots, \lambda_d)$ on the diagonal. Assume $\lambda_i > 0$ for all i (and hence, Λ^{-1} exists).

   a. The covariance matrix of $\widehat{w}$ is defined as $\Sigma = \mathbb{E}[(\widehat{w} - \mathbb{E}[\widehat{w}])(\widehat{w} - \mathbb{E}[\widehat{w}])^T]$. Argue that $\widehat{w} \sim \mathcal{N}(w_*, \Sigma)$ using properties about the transformation of Gaussian random variables as a given.

   b. Show that $\Sigma = \sigma^2(\mathbf{X}^T\mathbf{X})^{-1}$

   c. What is Σ in terms of $\mathbf{V}, \Lambda, \sigma^2$?

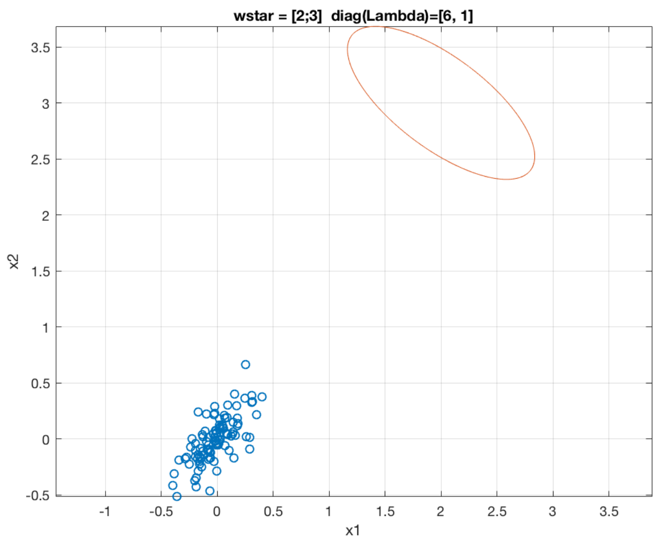
   d. What is the distribution of $\Lambda^{1/2}\mathbf{V}^T(\widehat{w} - w_*)$? What is $\mathbb{E}[||\Lambda^{1/2}\mathbf{V}^T(\widehat{w} - w_*)||_2^2]$?

   e. Define the confidence ellipsoid as $\mathcal{E}(A) = \{u \in \mathbb{R}^d : \sqrt{u^T A u} \le 1\}$. Argue that $||\Lambda^{1/2}\mathbf{V}^T(\widehat{w} - w_*)||_2^2 \le \nu^2$ if and only if $\widehat{w} - w_* \in \nu\mathcal{E}(\mathbf{X}^T\mathbf{X})$. Here, for any $A \subseteq \mathbb{R}^d$ and $c \ge 0$ we define $cA := \{ca : a \in A\}$.

   f. Let $v_1 = \begin{bmatrix} 3/5 \\ 4/5 \end{bmatrix}$ and $v_2 = \begin{bmatrix} -4/5 \\ 3/5 \end{bmatrix}$, $\mathbf{V} = [v_1,\ v_2]$, $\Lambda = \text{diag}(\lambda_1,\ \lambda_2)$, $\sigma^2 = 1$, $n = 100$. Different values of w_* and λ_1, λ_2 are given below. For each: draw $\widetilde{x}_i \sim \mathcal{N}(0, I_2)$ for $i = 1, \dots, n$, find an $A \in \mathbb{R}^{d \times d}$, $b \in \mathbb{R}^d$ such that $z_i = A\widetilde{x}_i + b$, $\sum_{i=1}^n z_i = 0$, and $\sum_{i=1}^n z_i z_i^T = \mathbf{V}\Lambda\mathbf{V}^T$. Plot z_i for $i = 1, \dots, n$ as well as the boundary of the confidence ellipsiod centered at w_* (Hint: consider $u = \mathbf{V}\Lambda^{-1/2}[\cos(\theta), \sin(\theta)]^T$ for $\theta \in [0, 2\pi]$).

      (a) $w_* = [3, 4]^T$, $\lambda_1 = 1$, $\lambda_2 = 1$
      (b) $w_* = [3, 4]^T$, $\lambda_1 = 8$, $\lambda_2 = 1$
      (c) $w_* = [3, 4]^T$, $\lambda_1 = 1$, $\lambda_2 = 8$
      (d) $w_* = [-4, 3]^T$, $\lambda_1 = 8$, $\lambda_2 = 1$
      (e) $w_* = [0, 4]^T$, $\lambda_1 = 8$, $\lambda_2 = 1$

Here is an example of what your figures should look like for $w_* = [2, 3]^T$ and $\lambda_1 = 6$, $\lambda_2 = 1$.



2 Kernel Regression

2. *[6 points]* First let's generate some data. Let $n = 30$ and $f(x) = 4\sin(\pi x)\cos(6\pi x^2)$. For $i = 1, \dots, n$ let each x_i be drawn uniformly at random on $[0, 1]$ and $y_i = f(x_i) + \epsilon_i$ where $\epsilon_i \sim \mathcal{N}(0, 1)$. Using regularized least-squares kernel regression, build a predictor

$$\widehat{\alpha} = \min_\alpha ||K\alpha - y||^2 + \lambda \alpha^T K \alpha \ , \qquad \widehat{f}(x) = \sum_{i=1}^{n} \widehat{\alpha}_i k(x_i, x)$$

where $K_{i,j} = k(x_i, x_j)$ is a kernel evaluation and λ is the regularization constant.

a. Using leave-one-out cross validation, find a good λ and hyperparameter settings for the following kernels:

- $k_{poly}(x, z) = (1 + x^T z)^d$ where $d \in \mathbb{N}$ is a hyperparameter,
- $k_{rbf}(x, z) = \exp(-\gamma \|x - z\|^2)$ where $\gamma > 0$ is a hyperparameter[1].

Record the values of d, γ, and the λ values for both kernels.

b. For a single plot per kernel, plot the original data $\{(x_i, y_i)\}_{i=1}^n$, the true $f(x)$, the $\widehat{f}(x)$ found through leave-one-out CV.

c. Suppose m additional samples are drawn i.i.d. the same way the first n samples were drawn. Propose a statistical significance test to decide which learned function (which kernel) is the better fit.

d. (Extra credit: *[3 points]*) Using the fixed hyperparameters you found in part a, we wish to build Bootsrap percentile confidence intervals for $\widehat{f}_{poly}(x)$ and $\widehat{f}_{rbf}(x)$ for all $x \in [0, 1]$. Use the non-parametric bootstrap with $B = 300$ datasets (i.e. randomly draw with replacement n samples from $\{(x_i, y_i)\}_{i=1}^n$ and train an $\widehat{f}$, repeat this B times) and find 5% and 95% percentiles (see Hastie, Tibshirani, Friedman Ch. 8.2 for a review). Plot the precentile curves on the plots from part b.

3 k-means clustering

3. *[5 points]* Given a dataset $x_1, ..., x_n \in \mathbb{R}^d$ and an integer $1 \le k \le n$, recall the following k-means objective function

$$\min_{\pi_1, \dots, \pi_k} \sum_{i=1}^{k} \sum_{j \in \pi_i} \|x_j - \mu_i\|_2^2 \ , \quad \mu_i = \frac{1}{|\pi_i|} \sum_{j \in \pi_i} x_j \ . \tag{1}$$

[1]Given a dataset $x_1, \dots, x_n \in \mathbb{R}^d$, a heuristic for choosing γ is the inverse of the median of all $\binom{n}{2}$ squared distances $||x_i - x_j||_2^2$.

Above, $\{\pi_i\}_{i=1}^k$ is a partition of $\{1, 2, ..., n\}$. The objective (1) is NP-hard[2] to find a global minimizer of. Nevertheless the commonly used heuristic which we discussed in lecture, known as Lloyd's algorithm, typically works well in practice. Implement Lloyd's algorithm for solving the k-means objective (1). Do not use any off the shelf implementations, such as those found in `scikit-learn`.

a. Run the algorithm on MNIST with $k = 5, 10, 20$, plotting the objective function (1) as a function of iteration. Visualize the cluster centers as a 28×28 image.

b. (Extra credit: *[3 points]*) Implement the `kmeans++` initialization scheme[3] for your k-means implementation. Note that this initialization scheme is widely used in practice, and as a rule should be used. Plot the objective function as a function of iteration.

4 Joke Recommender System

4. *[7 points]* You will build a personalized joke recommender system. There are $m = 100$ jokes and $n = 24,983$ users[4]. As historical data, every user read a subset of jokes and rated them. The goal is to recommend more jokes, such that the recommended jokes match the individual user's sense of humor. The historical rating is represented by a matrix $R \in \mathbb{R}^{n \times m}$. The entry $R_{i,j}$ represents the user i's rating on joke j. The rating is a real number in $[-10, 10]$: a higher value represents that the user is more satisfied with the joke. The directory `/jokes` contains the text of all 100 jokes. Read them before you start! In addition, you are provided with two files:

- `train.txt` contains the joke-user-score data representing the training set. Each line takes the form "`i, j, s`", where `i` is the user index, `j` is the joke index, and `s` is the user's score in $[-10, 10]$ describing how much they liked the joke (higher is better).
- `test.txt` has the same format, with the same users rating movies held out from the training set.

Latent factor model is the state-of-the-art method for personalized recommendation. It learns a vector representation $u_i \in \mathbb{R}^d$ for each user and a vector representation $v_j \in \mathbb{R}^d$ for each joke, such that the inner product $\langle u_i, v_j \rangle$ approximates the rating $R_{i,j}$. You will build a simple latent factor model. We will evaluate our learnt vector representations by two metrics

- Mean squared error: $\frac{1}{|S|} \sum_{(i,j) \in S} (\langle u_i, v_j \rangle - R_{ij})^2$ where S (and the corresponding $R_{i,j}$ values) are from the test set
- Mean absolute error: $\frac{1}{n} \sum_{i=1}^n \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} |\langle u_i, v_j \rangle - R_{ij}|$ where $\mathcal{N}_i$ are the jokes rated by user i in the test set

You will implement multiple estimators and use the inner product $\langle u_i, v_j \rangle$ to predict if user i likes joke j in the test data. You will choose hyperparameters like d or the amount of regularization by creating a validation set from the training set.

a. The first estimator pools all the users together and just predicts what the average user in the training set rated the joke. This is equivalent to $d = 1$ with u as the all ones vector and v minimizing least squares.

b. Now replace all missing values in $R_{i,j}$ no in the training set by zero. Then use singular value decomposition (SVD) to learn a lower dimensional vector representation for users and jokes. Recall this means to project the data vectors to lower dimensional subspaces of their corresponding spaces, spanned by singular vectors. Refer to the lecture materials on SVD, PCA and dimensionality reduction. You should use an efficient solver, I recommend `scipy.sparse.linalg.svds`. Try $d = 1, 2, 5, 10, 20$. Plot the error metrics on the train and test as a function of d?

[2]To be more precise, it is both NP-hard in d when $k = 2$ and k when $d = 2$. See the references on the wikipedia page for k-means for more details.

[3]See `http://ilpubs.stanford.edu:8090/778/1/2006-13.pdf`.

[4]Data from (removed)

c. For sparse data, replacing all missing values by zero is not a completely satisfying solution. A missing value means that the user has not read the joke, but doesn't mean that the rating should be zero. A more reasonable choice is to minimize the MSE only on rated joke. Let's define a loss function:

$$L\Big(\{u_i\}, \{v_j\}\Big) := \sum_{(i,j)\in T} (\langle u_i, v_j\rangle - R_{i,j})^2 + \lambda \sum_{i=1}^{n} \|u_i\|_2^2 + \lambda \sum_{j=1}^{m} \|v_j\|_2^2,$$

where T and $R_{i,j}$ here are from the training set and $\lambda > 0$ is the regularization coefficient. Implement an algorithm to learn vector representations by minimizing the loss function $L(\{u_i\}, \{v_j\})$. Note that you may need to tune the hyper-parameter λ to optimize the performance.

Hint: you may want to employ an alternating minimization scheme. First, randomly initialize $\{u_i\}$ and $\{v_j\}$. Then minimize the loss function with respective to $\{u_i\}$ by treating $\{v_j\}$ as constant vectors, and minimize the loss function with respect to $\{v_j\}$ by treating $\{u_i\}$ as constant vectors. Iterate these two steps until both $\{u_i\}$ and $\{v_j\}$ converge. Note that when one of $\{u_i\}$ or $\{v_j\}$ is given, minimizing the loss function with respect to the other part has closed-form solutions.