Homework 2

TAs: Baihong, Glenn, and Pierce

10-405/605: Machine Learning with Large Datasets

Due Monday, February 10th at 11:59 PM Eastern Time

Submit your solutions via Gradescope.

IMPORTANT: Be sure to highlight where your solutions are for each question when submitting to Gradescope otherwise you will be marked 0 and will need to submit regrade request for every solution un-highlighted in order for fix it!

Note that Homework 2 consists of two parts: this written assignment and a programming assignment. Remember to fill out the collaboration section found at the end of this homework as per the course policy.

Students enrolled in **10-605** are required to complete the following sections:

1. Written Section 1.1 *[24 points]*
2. Written Section 1.2 *[21 points]*
3. Programming Section 2 *[35 points]*

For students enrolled in **10-405**, the following modifications apply:

1. You are **NOT** required to complete **1.1.3 Kernel Version**.
2. You do **NOT** need to **compare exact kernel ridge regression** in **1.1.4**.

Your homework score will be calculated as a percentage of the maximum possible points, which is 80 for students in 10-605 and 70 for students in 10-405.

1 Written Section [45 Points for 10-605, 35 for 10-405]

1.1 Regression, regression, regression [24 Points for 10-605, 14 for 10-405]

In this problem you will learn a regression model with the same dataset in several different ways. You may use any mix of manual calculation and computer code that you like. (You should only need short segments of code.) If you use any code, please include your code in the text of your written (part 1) submission — not as part of the programming submission.

The dataset for the regressions is below:

x_1	x_2	const	y
1	1	1	1
1	0	1	2
0	1	1	1
2	1	1	-1

For each of the parts 1.1.1–1.2 below, you will need to solve a system of linear equations. Please be sure to include the **actual numerical equations** that you solve in the answer to your question, i.e., write out the matrix A and the vector b if you solve $Ax = b$. Hint: the function `numpy.linalg.lstsq` can be very helpful for solving systems of linear equations, including if you need to find the minimum norm solution. But please *do not* use its least squares functionality: i.e., if you want to solve a least squares problem, do it by constructing a set of linear equations that we can solve with zero residual error.

1.1.1 Exact linear regression *[5 points]*

Solve the linear regression problem for this dataset exactly by computing the closed form solution (using the covariance matrix). Be sure to report the numerical matrix and vector for the normal equations, as well as the learned weight vector.

```
A = np.array([
[1, 1, 1],
[1, 0, 1],
[0, 1, 1],
[2, 1, 1]
])
b = np.array([1, 2, 1, -1])
ATA = A.T @ A
ATb = A.T @ b
ATA_inv = np.linalg.inv(ATA)
w = ATA_inv @ ATb
```

——output——

A$^T A$: [[634][333][434]]

$A^T b$: [113]

$(A^T A)^(-1)$: [[5.00000000e−01−7.40148683e−17−5.00000000e−01][−1.11022302e−161.33333333e+00 − 1.00000000e + 00][−5.00000000e − 01 − 1.00000000e + 001.50000000e + 00]]

Learned weight vector w: [-1, -1.66666667, 3]

1.1.2 Linear regression via gradient descent *[5 points]*

Suppose you instead use gradient descent to approximately solve the linear regression problem. What is the gradient descent update for the objective at iteration $i+1$? The update should depend on the previous model iterate, $\mathbf{w}_i$; the step size, α; and the dataset provided above. If $\mathbf{w}_i = \begin{pmatrix} -2 \\ -3 \\ 1 \end{pmatrix}$ and $\alpha = 0.1$, what will be the value of $\mathbf{w}_{i+1}$?

```
A = np.array([
      [1, 1, 1],
      [1, 0, 1],
      [0, 1, 1],
      [2, 1, 1]
])
b = np.array([1, 2, 1, -1])
w_i = np.array([-2, -3, 1])
grad = (1 / A.shape[0]) * (A.T @ (A @ w_i - b))
alpha = 0.1
w_next = w_i - alpha * grad
```

——output——

Updated weight vector w_{i+1} : $[-1.55, -2.675, 1.4]$

1.1.3 Kernel version (10-605 only) *[8 points]*

Now switch to using the squared-exponential kernel,

$$k(x, x') = \exp(-\tfrac{1}{2}\|x - x'\|^2)$$

Solve the kernel ridge regression problem exactly using the Gram matrix form, using ridge parameter $\lambda = 1$. Be sure to report the numerical matrix and vector for the regression equations, as well as the resulting example weight vector.

Make a prediction of the label for the following new point: $x_{\text{new}} = (0, 0, 1)$. Report the formula that you use to make the prediction (including numerical values for all quantities) as well as the final prediction.

```
def squared_exponential_kernel(x1, x2):
    return np.exp(-0.5 * np.linalg.norm(x1 - x2) ** 2)
X_train = np.array([
    [1, 1, 1],
    [1, 0, 1],
    [0, 1, 1],
    [2, 1, 1]
])
y_train = np.array([1, 2, 1, -1])
m = X_train.shape[0]
K = np.zeros((m, m))
for i in range(m):
    for j in range(m):
        K[i, j] = squared_exponential_kernel(X_train[i], X_train[j])
lambda_reg = 1
I = np.eye(m)  # Identity matrix
alpha = np.linalg.inv(K + lambda_reg * I) @ y_train
x_new = np.array([0, 0, 1])
K_new = np.array([squared_exponential_kernel(x_new, X_train[i]) for i in range(m)])
y_new = K_new @ alpha
```

```
——output——
Gram Matrix (K): [[1. 0.60653066 0.60653066 0.60653066] [0.60653066 1. 0.36787944 0.36787944] [0.60653066
0.36787944 1. 0.13533528] [0.60653066 0.36787944 0.13533528 1. ]]
Alpha values: [ 0.36634729 0.98992087 0.26168507 -0.81089381]
Kernel vector for new point: [0.36787944 0.60653066 0.60653066 0.082085 ]
Predicted value for x_new: 0.8273467967630496
```

1.1.4 When and why? 10-605: *[6 points]* , 10-405: *[4 points]*

For each of the methods that you used above (exact covariance-form regression, gradient descent, and exact kernel ridge regression), please give a few sentences saying when we might use this method and why. (For students in 10-405, you do not need to consider exact kernel ridge regression.)

1. exact convariance-form regression:
When:
small to medium-sized datasets where matrix inversion is feasible
when the features are linear
when we need an exact solution rather than approximation
Why:
fast for low-dimension data and exact solution

2. gradient descent:
When:
when dealing with large datasets
high-dimensional features
approximate solution is acceptable
Why:
unlike closed-form solutions, gradient descent can handle millions of features

3. exact kernal ridege regression:
When:
dealing with nonlinear data
when we want to apply k
approximate solution is acceptable
Why:
By using a kernel function, it can model complex relationships in the data

1.2 Hash Kernels [21 Points]

This part of the question refers to the paper "Hash Kernels" by Shi et al, PMLR 5:496-503, 2009, available on-line at https://proceedings.mlr.press/v5/shi09a/shi09a.pdf. These questions may require you to make some assumptions about what was done—if you are unsure state your assumptions in your answer.

1.2.1 Degradation due to hash collisions *[2 points]*

In one experiment of the paper, the authors evaluated how small a hash table needed to be, at reported how the size of the hash table affected collision rate, and the accuracy of the final classifier. What table gives these results, which dataset did they use for evaluation, and what is the smallest hash table for which accuracy was as good as accuracy with the 22-bit hash table?

1. Table 3
2. RCV1
3. 18 bits

1.2.2 Compression achievable with hashing *[2 points]*

How many buckets are in that smallest hash table, and how much smaller is it than the 22-bit hash table?

$\frac{2^{14}}{2^{22}} = \frac{1}{256}$

1.2.3 Analysis of bias 1 *[2 points]*

In the paper Section 4 compares the values of $\overline{k}^h(x, x')$ and $k(x, x')$. In your own words, what are these functions? One sentence each should be enough.

$\overline{k}^h(x, x')$ is an approximation of $k(x, x')$ obtained using a hash-based method.
$k(x, x')$ is the true kernel function that computes exact similarity of x and x'

1.2.4 Analysis of bias 2 *[5 points]*

Assume that instances are documents, and all the feature weights are non-negative. What is the relationship between $k(x, x')$ and $\overline{k}^h(x, x')$? Explain your answer briefly.

1. It is always true that $k(x, x') \geq \overline{k}^h(x, x')$.
2. It is always true that $k(x, x') \leq \overline{k}^h(x, x')$.
3. It is always true that $k(x, x') = \overline{k}^h(x, x')$.

Only It is always true that $k(x, x') \geq \overline{k}^h(x, x')$ is correct.
The approximated kernel $k^h(x, x')$ is derived using a hashing method, which introduces collisions, causing some distinct features to be mapped to the same bucket. This leads to underestimation of the true kernel similarity.

1.2.5 Joint hashing of labels and features *[5 points]*

Assume that HLF hash kernels are used with a binary classifier f (e.g., a logistic regression classifier), and that the HLF hashing method used follows Equation 8 rather than the optimization described below it. Describe a plausible process for using HLF in a multiclass setting where there are M disjoint class labels $y_1, \ldots, y_M$, by giving pseudocode describing (a) how the training data $(x_1, y_1) \ldots, (x_N, y_n)$ is preprocessed and used to train f, and (b) the method used at test time to classify single a document vector x^*.

```
# Input: Training data (x1, y1), ..., (xN, yN)
# Output: Trained classifier f using HLF representation

(a) training process
for each (x,y) in training_data:
    hashed_representation = {}
    for feature in x:
        h_key = hash_function(feature, y)
        hashed_representation[h_key] += 1
    f.train(hashed_representation,y)

(b) testing process(classification)
for each class y in possible_classes:
    hashed_representation = {}

    for feature in x*:
        h_key = hash_function(feature, y)
        hashed_representation[h_key] += 1

    score_table[y] = f.predict(hashed_representation)

return argmax(score_table)
```

1.2.6 Approximation of DF *[5 points]*

Give pseudocode for the process used by the authors to approximate DF in the hash space, assuming there are B buckets.

```
#Step 1: Process each document and update DF counts
for document in D:
    unique_terms = set(document)  # Extract unique terms in the document

    for term in unique_terms:
        bucket_index = hash_function(term) % B  # Compute hash bucket
        DF_hash[bucket_index] += 1  # Increment DF for that hash bucket

#Step 2: DF_hash now contains approximated document frequencies
return DF_hash
```

2 Programming [35 Points]

2.1 Linear Regression on the Million Songs Dataset [35 Points]

2.1.1 Introduction

In this section, you will train a linear regression model to predict the release year of a song given a set of audio features.

We provide the code template for this section assignment in *one* Jupyter notebook. What you need to do is to follow the instructions in the notebooks and implement the missing parts marked with '`<FILL_IN>`' or '`# YOUR CODE HERE`'. Most of the '`<FILL_IN>/YOUR CODE HERE`' sections can be implemented in just one or two lines of code. Also, keep in mind to delete the '`raise NotImplementedError()`' once you are done implementing a function. We provide several `assert` statements in the notebook for you to check the validity of your solution. Finally, note that the presence or location of comments in the cell (such as '`# YOUR CODE HERE`') do not affect the autograder.

2.1.2 Getting Started

1. **Getting lab files**
   You can obtain the notebook `'hw2_coding.ipynb'` for this programming questions in .zip file.

   Next, import the notebook into your Databricks account, which provides you a well-configured Spark environment and will definitely save your time (see Homework 1 handout for details).

2. **Writing your code**
   In order to enable auto-grading, please do not change any function signatures (e.g., function name, parameters, etc) or delete any cells. If you do delete any of the provided cells (even if you re-add them), the autograder will fail to grade your homework. If you do this, you will need to re-download the homework files and fill in your answers again and resubmit.

3. **Preparing for submission**
   We provide several public tests via `assert` in the notebook. You may want to pass all those tests before submitting your homework.

4. **Submission**
   (a) Export the solution notebooks as IPython notebook files on Databricks via `File -> Export -> IPython Notebook`
   (b) Submit the completed notebook via Gradescope.

2.1.3 Instructions

This section covers a common supervised learning pipeline, using a subset of the Million Song Dataset from the UCI Machine Learning Repository. Our goal is to train a linear regression model to predict the release year of a song given a set of audio features.

In this section, you will be implementing a common supervised learning pipeline, using a subset of the Million Song Dataset from the UCI Machine Learning Repository, to train a linear regression model to predict the release year of a song given a set of audio features.

In this part, we will cover

- Part 1: Reading and parsing the Million Song dataset
- Part 2: Creating and evaluating a baseline model
- Part 3: Training (via gradient descent) and evaluating a linear regression model
- Part 4: Training using SparkML and tune hyperparameters via grid search
- Part 5: Adding interactions between features

See the notebook for detailed descriptions and instructions of each question.

2.1.4 Deliverables

The deliverable for this programming section is the completed .ipynb notebook. Please submit your notebook to the autograder on Gradescope. Note that for Homework 2 there is only a single (*one*) autograded notebook. You score for section 2.1 will be entirely determined by the score given by the autograder.

Make sure you do not forget to complete deliverables for all programming sections. In this homework not all programming sections will be autograded, so a full score on the autograder does not mean you have completed all programming assignments.

3 Collaboration Questions

1. (a) Did you receive any help whatsoever from anyone in solving this assignment?

   no

   (b) If you answered ‘yes’, give full details (e.g. “Jane Doe explained to me what is asked in Question 3.4”)

2. (a) Did you give any help whatsoever to anyone in solving this assignment?

   no

   (b) If you answered ‘yes’, give full details (e.g. “I pointed Joe Smith to section 2.3 since he didn’t know how to proceed with Question 2”)

3. (a) Did you find or come across code that implements any part of this assignment?

   no

   (b) If you answered ‘yes’, give full details (book & page, URL & location within the page, etc.).