CS 145 PS1

September 29, 2016

Instructions / Notes:

- Using the Jupyter version of this problem set is **strongly recommended**, however you can use only this PDF to do the assignment, or replicate the functionality of the Jupyter version by using this PDF + your own SQLite interface
- Note that the problems reference tables in the SQLite database (in the PS1.db file) however solution queries must be for any general table of the specified format, and so use of the actual database provided is *not necessary*
- See Piazza for submission instructions
- Have fun!

Problem 1: Matrix Manipulations

Two random 3x3 ($N = 3$) matrices have been provided in tables A and B, having the following schema:

```
i   INT:  Row index
j   INT:  Column index
val INT:  Cell value
```

Note: all of your answers below must work for any square matrix sizes, i.e. any value of N. Note also how the matrices are represented- why do we choose this format?

Part (a): Elementwise multiplication

We want the *elementwise multiply* of matrices A and B. That is, $(A \cdot B)_{ij} = A_{ij}B_{ij}$ for all i, j.

Write a *single SQL query* that performs this operation (in the same format as A- output tuples should be of format (i, j, val) where i is row, j is column, and the output is ordered by row then column index). If executed on the tables A and B in PS1.db, the result should be:

```
i j val
0 0 63
0 1 30
0 2 80
1 0 70
1 1 42
1 2 63
2 0 2
2 1 0
2 2 35
```

Part (b): Permutations

A *permutation* is a mapping from a set of integers to itself, i.e. $\pi : [n] \to [n]$, where $[n]$ denotes the set $\{0, \ldots, n-1\}$. We write:

$$\pi([a_1, a_2, \ldots, a_n]) = [a_{\pi(1)}, a_{\pi(2)}, \ldots, a_{\pi(n)}]$$

The provided table c contains a permutation on [3].

```
ind pi
0   2
1   0
2   1
```

Write a *single SQL query* that permutes the rows of matrix A according to the permutation stored in c.

$$(i, j, A_{ij}) \to (i, j, A_{\pi(i)j})$$

For the example provided, this means a single-step cyclic upward shift, however your code needs to work for any input A, c of the right dimensions. If executed on the tables A and c in PS1.db, the result should be:

```
i j val
0 0 10
0 1 7
0 2 7
1 0 2
1 1 0
1 2 5
2 0 7
2 1 5
2 2 8
```

Part (c): Composability of permutations

A known property of permutations, is that they are closed under composition. This implies that applying two permutations in succession is equivalent to applying a different, single permutation.

$$\pi_2(\pi_1([a_1, a_2, \ldots, a_n])) = [a_{\pi(1)}, a_{\pi(2)}, \ldots, a_{\pi(n)}]$$

where $\pi(i) = \pi_2(\pi_1(i))$.

Write a *single SQL query* that applies the permutation in table c twice on the rows of matrix A.

$$(i, j, A_{ij}) \to (i, j, A_{\pi(\pi(i))j})$$

For the example provided, this means a two-step cyclic upward shift, however your code needs to work for any input A, c of the right dimensions. If executed on the tables A and c in PS1.db, the result should be:

```
i j val
0 0 2
0 1 0
```

```
0 2 5
1 0 7
1 1 5
1 2 8
2 0 10
2 1 7
2 2 7
```

Part (d): Local maximum

Consider the *local maximum* matrix function, $MAX : \mathbb{R}^{n,n} \to \mathbb{R}^{n,n}$

$$MAX(A)_{ij} = \max\{A_{ij} \text{ and cells up, down, left, and right}\}$$

that is, if

$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

then

$$MAX(A)_{00} = \max\{a, b, d\}$$
$$MAX(A)_{10} = \max\{a, d, g, e\}$$

and

$$MAX(A)_{11} = \max\{d, b, e, h, f\}$$

Write a *single SQL query* that computes the local maximum of matrix A. If executed on table A in PS1.db, the result should be:

```
i j val
0 0 10
0 1 8
0 2 8
1 0 10
1 1 10
1 2 8
2 0 10
2 1 7
2 2 7
```

Problem 2: U.S. Sustainable Energy Sources

We've prepared and loaded a public dataset (https://catalog.data.gov/dataset/energy-generation-by-state-and-technology-2009-69f4f) from the US DOE (Department of Energy) of sustainable energy production in MWh (megawatt hours, i.e., 1000 kilowatt hours). This data is from 2009, the latest year available. The data includes each state, the region of the United States it is in, and its production in MWh of sustainable energy by source (solar, wind, hydroelectric, and nuclear). The table energy has the following schema:

```
TABLE energy (
    state varchar(30),
    region varchar(30),
    solar float,
    wind float,
    hydro float,
    nuclear float)
```

Part (a): State Champions

Using a *single SQL query*, find all of the regions in the United States with a state in it that is the leading producer of one of the four types of energy (solar, wind, hydro, and nuclear), and return the counts of how many state winners they had in descending order. **Do not include any regions with no state winners.**

Further requirements:

- Use GROUP BY
- Write the shortest possible SQL query to accomplish this
- Return relation (region, num_state_winners)

If executed on the energy table in PS1.db, the result should be:

```
region      num_state_winners
West        2
Midwest     1
Southeast   1
```

Part (b): Pareto Frontiers

Solar power and wind power tend to be complementary, since it tends to be less windy when there are few clouds and the sun can best reach solar panels.

Our goal in this part is to identify states that strike the best balance between solar and wind energy production. Here we define a state as "best" if it exists on the Pareto frontier of solar and wind energy production. In other words, a state is Pareto optimal if no other state produces both more solar and more wind energy, and the Pareto frontier is the set of states that are Pareto optimal.

Write a query that returns the entire Pareto frontier. Results should be triples of the form $(state, solar, wind)$, where *state* is the name of the state in the frontier, and *solar* and *wind* are its solar and wind energy production in MWh. Order the results in descending order by sum total of solar and wind energy production in MWh.

If executed on the energy table in PS1.db, the result should be:

```
state        solar        wind
Texas        0.0          19367238.86
California   611763.387   5764637.309
```

Part (c)

Find a list of regions which had a minimum state nuclear power *greater than 10% of the maximum state nuclear power value* (where *minimum state nuclear power* = minimum value of nuclear power production over all states with non-zero nuclear production).

Note: do not hard-code the maximum state nuclear power or any other input-data-dependent values.

Do this using GROUP BY and aggregate functions (e.g. COUNT, SUM, MAX, MIN).

If executed on the energy table in PS1.db, the result should be:

```
region
Mid Atlantic
Southeast
```

Part (d)

Do the same as above, except do **not** use GROUP BY or any aggregate functions.

Problem 3: Classification

SQL is a very expressive language (some SQL extensions are known to be Turing complete). Here, we go through the steps of using it for a simple machine learning task: *binary classification.*

We will use a subset of the Iris dataset. The reduced dataset included here as table IRIS, is comprised of 100100 samples with features SepalLength, SepalWidth, PetalLength, PetalWidth and labels 0 (identifying the species Iris-setosa) and 1 (the species Iris-versicolor). Samples of the third class (Iris-virginica) present in the original dataset have been dropped for this exercise.

Model

We will use a simple *linear model* to predict the label (species) given the four features included in the dataset. An unlabeled sample $x_i \in \mathbb{R}^4$ is a vector of the four given feature values. For example, the first sample in our dataset is $x_0 = [5.1, 3.5, 1.4, 0.2]^\top$. This model, known as the Perceptron, is also represented as a four-dimensional vector $w \in \mathbb{R}^4$. Given sample x_i and model w the perceptron makes the following prediction $\hat{y}_i$ for the true label y_i of sample x_i:

$$\hat{y}_i = \begin{cases} 1 & \text{if } x_i^\top w > 0 \\ 0 & \text{otherwise} \end{cases}$$

A pre-trained model, w is included in table MODEL:

```
j  val
0  0.35260621
1  -0.90142873
2  0.59729474
3  1.30194557
```

Part (a): Matrix-vector Multiplication

If we consider the *feature matrix*, X whose i-th row is $x_i^\top$, the prediction rule for all samples becomes:

$$\hat{y} = \text{step}(Xw)$$

where Xw is a matrix-vector multiplication and

$$\text{step}(z)_i = \begin{cases} 1 & \text{if } z_i > 0 \\ 0 & \text{otherwise} \end{cases}.$$

The final product, $\hat{y} \in \{0,1\}^n$ is a vector with the predictions for all samples.

The product of a matrix X (having dimensions $n \times m$) and a vector w (having dimensions $m \times 1$) is the vector c (of dimension $n \times 1$) having cell at row i and column j equal to:

$$c_i = \sum_{j=1}^{m} X_{ij} w_j$$

In other words, to do matrix-vector multiplication, get each cell of the resulting vector c, c_i, by taking the *dot product* of the ith row of X and w.

We start by preprocessing IRIS to create the X feature matrix, using the following schema:

```
i   INT:  Row index
j   INT:  Column index
val INT:  Cell value
```

In order to streamline the work in this section we will make use of views. A view is a virtual table based on the output set of a SQL query. A view can be used just like a normal table; the only difference under the hood is that the DBMS re-evaluates the query used to generate it each time a view is queried by a user (thus the data is always up-to date!)

```
DROP VIEW IF EXISTS X;
CREATE VIEW X AS
SELECT i, 0 as j, SepalLength as val
FROM IRIS
UNION
SELECT i, 1 as j, SepalWidth as val
FROM IRIS
UNION
SELECT i, 2 as j, PetalLength as val
FROM IRIS
UNION
SELECT i, 3 as j, PetalWidth as val
FROM IRIS;
```

Write a single SQL statement that implements the matrix-vector multiply Xw between the provided view X and model w from table MODEL. Return the first 5 tuples as your answer. If executed on the data provided in PS1.db, the result should be:

```
i  val
0  -0.260107134
1  0.120085989
2  -0.190450473
3  -0.016109273
4  -0.385510628
```

Part (b): Predict labels

Now we can predict the labels using the following rule:

$$\hat{y} = \text{step}(Xw)$$

Create a view named 'PREDICTION' that will produce tuples of sample IDs and label predictions, i.e. $(i, \hat{y}_i)$.

Not sure you got the previous question right? The provided view 'ANSWER_P3a' with schema $(i, (Xw)_i)$ has the right answer. You can use it in your solution here for full credit in this question. **Warning**: Using this view in Part (a) will result in zero credit for that problem.

Part (c): Evaluate Accuracy

Given the predicted labels $\hat{y}_i$ and true labels y_i we evaluate the predictive power of our model using *accuracy*: the fraction of labels our model got right.

$$\text{accuracy} = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}[y_i = \hat{y}_i]$$

Using the 'PREDICTION' view from the previous part should make for a simpler solution.

Not sure you got the previous question right? The provided view 'ANSWER_P3b' with schema $(i, \hat{y}_i)$ has the right answer. You can use it in your solution here for full credit in this question. **Warning**: Using this view in Part (b) will result in zero credit for that problem.

If executed on the data provided in PS1.db, the result should be:

```
accuracy
0.87
```

Our pre-trained classifer achieved a classification accuracy of 87%. If you don't think this is good enough, you are correct. This is not an optimally trained model. Try the bonus question to see how this classification performance can be improved!

Bonus Problem 1: Classification, Pt. II

A simple procedure from training the model w given labeled data, is as follows.

$$w' = w + 0.0001\sum_{i=1}^{n}(y_i - \hat{y}_i)x_i$$

This kind of optimization algorithm is typically applied iteratively, but here we will just run a single step.

Run a SQL statement that computes the new model value, w', based on the original model w in MODEL and the samples x_i in IRIS.

Using answers from previous parts. As before, you should feel free to use the provided views 'ANSWER_P3a' and 'ANSWER_P3b' described in parts (b) and (c) in your solution for full credit in this question. **Warning**: Using these views in Parts (a) and (b) respectively will result in zero credit for that problem.

If executed on the data provided in PS1.db, the result should be:

```
j  val
0  0.34618621
1  -0.90557873
2  0.59523474
3  1.30153557
```

We have a new model now. If you perform the evaluation process for this new model, you should find that accuracy has increased to 90%. Not bad for a single step of the algorithm! Further iterations will improve this performance more.