SOLUTION: DO NOT DISTRIBUTE

TBD
09:30 BST
Duration: 2 hours
Additional Time: 30 minutes
Timed exam — fixed start time

DEGREES of MSci, MEng, BEng, BSc, MA and MA (Social Sciences)

Introduction to Data Science and Systems ¶ COMPSCI 5089

(Answer All 4 Questions)

This examination paper is an open book, online assessment and is worth a total of 60 marks.

1. **(a)** You are designing an application for clothing shops to predict clothes size based on customer height and weight. Suppose we have a **clothing dataset** with height, weight and the corresponding T-shirt size of several customers.

customer ID	height	weight	size
U1	170	60	M
U2	172	60	M
U3	173	61	M
U4	173	64	L
U5	175	67	L
U6	175	66	L

You can represent this dataset based on their vector representations by regarding height and weight as two dimensions. Now there is a new client **Abel** (U0) whose height is 173cm and weight is 62kg. You are asked to predict the T-shirt size for Abel.

(i) Calculate the Euclidean distance (L2 Norm) between the new point and the existing points.

[3]

Solution:
0.5 mark for each correct calculation. Keeping other decimal places rather than 2 decimal places will not reduce their marks.
d(U0,U1)=3.61 **[0.5]**
d(U0,U2)=2.24 **[0.5]**
d(U0,U3)=1.00 **[0.5]**
d(U0,U4)=2.00 **[0.5]**
d(U0,U5)=5.39 **[0.5]**
d(U0,U6)=4.47 **[0.5]**

(ii) Predict the size of Abel, based on the kNN algorithm, with $k = 3$ and the above calculated distances. Justify your prediction.

[2]

Solution:
The three most closest neighbours are U2 (M), U3 (M) and U4 (L). **[1]**
So you should predict M as the T-shirt size for Abel. **[1]**

(b) *For all answers, include in your answer document both code and the output of that code.*

(i) Calculate the covariance matrix for the clothing dataset using numpy.

[1]

Solution:

```
cov = np.cov(users, rowvar=False)
# cov: array([[3.6, 5.2],
#             [5.2, 9.6]])
```

[1]

(ii) Calculate the eigenvector and eigenvalues the covariance matrix using numpy.

[2]

Solution:

```
evals, evecs = np.linalg.eig(cov)
# evals: array([ 0.59666759, 12.60333241])

# array([[-0.86594528, -0.50013875],
#        [ 0.50013875, -0.86594528]])
```

[2]

(iii) Dimensionality reduction. Map the **clothing dataset** into principal component with the largest eigenvalue of its covariance matrix.

[2]

Solution:
The largest eigenvalue is evals[1], so we should map data into evecs[:,1].

```
users@evecs[:,1]
# array([-136.98030494, -137.98058245, -139.34666648,
#         -141.94450232, -145.54261566, -144.67667038])
```

[2]

(c) (i) Find SVD for $A = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$, you should include full working in your solution.

[3]

Solution:

- Solution 1:

$A = U\Sigma V^T$.
$A^T A = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$ with eigenvalues $\lambda_1 = 1$ and $\lambda_2 = 0$ and corresponding eigenvectors $\mathbf{u}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $\mathbf{u}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$. Hence $U = \begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 \end{bmatrix}$ **[1]**
$\Sigma = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ **[1]**

$AA^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ with eigenvalues $\lambda_1 = 1$, $\lambda_2 = 0$, and $\lambda_3 = 0$ and corresponding eigenvectors $\mathbf{v}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ $\mathbf{v}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$ and $\mathbf{v}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$.
Hence $V^T = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \mathbf{v}_3 \end{bmatrix}$ **[1]**

- Solution 2:

  Another feasible solution is to use `np.linalg.svd` to solve this question.

```
a = np.array([[1,0],[0,0],[0,0]])
np.linalg.svd(a, full_matrices=True)
# u    [1., 0., 0.],
#      [0., 1., 0.],
#      [0., 0., 1.]
# s    [1,0]
#      [0,0]
#      [0,0]
# vt   [1,0]
#      [0,1]
```

(ii) State the relations between determinant, matrix inversion and non-singular.

[2]

Solution:
The inverse of matrix exists only if its determinant value is a non-zero value **[1]**
A matrix is non-singular if the determinant is different from zero **[1]**

2. Consider a tennis player, Ed Balls, who wants to prepare for a competition match against an opponent—let's call him Frank Racket. In order to prepare for the match, Ed has acquired records of the 100 previous matches of his opponent and wants to study statistics of Frank's play to choose where to focus his training.

(Here is a quick summary of the rules of tennis: `https://protennistips.net/tennis-rules/`)

Ed is interested in studying Frank's serve as this can be an important strategic advantage.

- For a serve to be valid, it must pass the net and bounce in the diagonally opposite service box.
- If the first serve is a fault (eg, hits the net or bounces outside the service box), the player can attempt a second serve.
- If the player makes a second fault, he loses the point.

Ed wants to study where Frank's serve bounce in the service box to plan his positioning on the court. We have $N_F = 1,000$ examples of first serve from Frank, and $N_S = 1,000$ examples of second serve. We want to estimate the distributions of the bounce location $\mathbf{x}$ for Frank's first $p(\mathbf{x}|first)$ and second serves $p(\mathbf{x}|second)$.

For simplicity,

- we denote the corner closer to the net and towards the centre of the court as position (0,0), and the corner towards the outside of the court and away from the net as (1,1).
- We will ignore serves that hit the net

This means that values outside $[0,1] \times [0,1]$ indicate that the serve is a fault.

(a) How would you use the *empirical distribution* to get an estimate of $p(\mathbf{x}|first)$? Explain the steps, the parameters that need to be set and the associated trade-offs.

[4]

Solution:

(This was seen in lectures, but students need to apply it to this problem)

The empirical distribution is an approximation of a distribution with a normalised histogram. **[0.5]**

1. divide the serve box horizontally and vertically in B bins, creating a total of $B \times B$ bins. **[0.5]**
2. record the number of serves in our dataset that bounced in each bin **[0.5]**
3. normalise the histogram by the total number of serves in the dataset **[0.5]**

The parameters to consider:

- The number of bins B **[0.5]** if set too small the estimate will be imprecise, if set too large the estimate will be noisy due to the limited number of examples in the dataset. **[0.5]**
- We need to handle serves outside the serve box (faults). **[1]**

(b) Ed now wants to model Frank's serves using a *normal distribution*:

$$f_X(x) = \frac{1}{Z} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

(i) Explain the parameters, their effect on the distribution and the best way to estimate them in this scenario.

[4]

Solution:
The parameters of the normal distribution are:

- the mean vector μ **[0.5]** the mean of the distribution **[0.5]**
- the covariance matrix Σ **[0.5]** determines the shape of the distribution, how wide, narrow or skewed it is **[0.5]**

For the normal distribution, we have *standard estimators*, closed forms estimates of the distribution parameters from the available samples. **[1]**

$$\mu = \frac{1}{N_F} \sum_i \mathbf{x_i}$$

[0.5]

$$\Sigma_{ij} = \frac{1}{N-1} \sum_{k=1}^{N} (X_{ki} - \mu_i)(X_{kj} - \mu_j)$$

[0.5]
(Note: This comes from the linear algebra lectures, this is all in the lectures but requires that the student tie together the 1D example in the probabilities lecture to the linear algebra lecture)

(ii) What could be the problem with this choice of model? Give an example of a situation where it would be inappropriate (you can use a diagram to illustrate your example).

[2]

Solution:
The main problem is if the data is *not* normally distributed—typically if it has more than one mode. For example if Frank serves half of his serves in the bottom left corner of the box and the other half in the bottom right corner of the box, then the normal distribution would estimate a mean in the centre bottom of the box, with a large variance. This is misleading as Frank never actually serves at this location, whereas the model would deem this as a high probability.
[1] for pointing the multiple mode issue. **[1]** for a valid example or diagram.

(c) Ed has found that his normal model is not accurate enough for him. In order to get a more accurate modelling of the data, he decides to use a *mixture of Gaussians* to model his data.

Explain how the model would be parameterised, and how you would fit the model to the available data (provide the relevant equations).

[5]

Solution:

A mixture model is defined as follows (from the lecture motes)

$$f_X(x) = \sum_i \lambda_i n_X(x; \mu_i, \Sigma_i)$$

Where n_X is the standard multivariate Gaussian. **[1]**

In this model the parameters are:

- The number of Gaussians in the model N **[0.5]**
- Then for each Gaussian:
- λ is a scalar, the relative weight of this Gaussian in the mixture **[0.5]**
- μ is the mean vector for this Gaussian **[0.5]**
- Σ is the covariance of this Gaussian **[0.5]**

N is fixed, so we want to fit the parameter vector $\theta = [\lambda_1, \mu_1, \Sigma_1, \ldots, \lambda_N, \mu_N, \Sigma_N]$

For a mixture of Gaussian we do not have standard estimators like for the normal distribution, so we need to use the log likelihood. **[1]**

$$\arg\min_\theta \sum_{i=1}^{N_F} \log \sum_{j=1}^{N} \lambda_j n_X(x_i; \mu_j, \Sigma_j)$$

[0.5]

This could be optimised using for example gradient descent or variants of the Newton method **[0.5]**

3. Pretend that you are the new head of a local radio, *IDSS Radio* being tasked with renewing the radio's image and programme. The radio's programming and popularity has varied over the years and you want to use a data science approach to find the right type of programming for the local audience. To this end you start by categorising the programming of the radio between types of content:

$$\mathscr{C} = \{music, news, business, fiction, comedy, advertisement\}$$

You have historical records of the proportion of each content type in the radio programme for every month over the last ten years, as well as a rating r by a sample of the audience on scale between 1 and 10, where 1 means "hate it" and 10 means "love it".

Considering a programme $\mathbf{p} = [p_m, p_n, p_b, p_f, p_c, p_a] \in \mathbb{R}^5$ that gives the number of hours for each content type, we are interested in studying the function $r(\mathbf{p})$ that gives the listeners' rating for this programme.

(a) As a first attempt, you decide to assume that the function $r(\mathbf{p})$ is linear, and therefore to solve it using linear-least-squares, of the canonical form (from the lecture notes):

$$\arg\min_{\mathbf{x}} L(\mathbf{x}) = \|A\mathbf{x} - y\|_2^2$$

(i) Explain what each variable in this equation means in this scenario, specifying their dimension, and what would be the result.

[4]

Solution:
In this case - $A = [p_m, p_n, p_b, p_f, p_c, P_a, 1]$ is the number of hours for each content type, a matrix of dimension $120 \times 6\$$ **[0.5]** where the first 5 columns correspond each to one type of content **[0.5]** the last one is set to 1 to allow for the constant in the linear relation. **[0.5]** Each row correspond to one month of historical data. **[0.5]** - $\mathbf{y}$ is a vector of dimension 120, recording the average rating for the radio station for each month on record. **[0.5]** - $\mathbf{x}$ is a matrix of parameters representing the linear relation between each content and the rating, a vector of dimension 1×6 (1×5 will also be accepted as already penalised above) **[0.5]** - The outcome will be the coefficient for the linear relation $\hat{\mathbf{x}}$ **[0.5]** that best fits the recorded data **[0.5]** such that

$$\hat{r}(\mathbf{p}) \triangleq [\hat{x}_1, \hat{x}_2, \hat{x}_3, \hat{x}_4, \hat{x}_5] \cdot \mathbf{p} + \hat{x}_6$$

(The last equation is not needed to get the marks).

(ii) Could you name a reason why this may not be a good model? How could you measure this using your data?

[3]

Solution:
The main reason for this model to fail is if *the relation is not linear* - which is likely to be the case. **[1]** You test this by looking at the *residuals*: the difference between the

ratings predicted by the model and actually recorded in the data **[1]** If the model is good they should be very small **[0.5]** whereas large values would indicate an example poorly modelled **[0.5]**

(b) We want to try and fit another model, this time assuming that viewers' preferences peak for certain quantities of each program, and then decreases again if the quantity increases even more. We could model this quantity preference as a bell shaped distribution function over the quantity p_z for each type of content z:

$$B_z(p_z) = \alpha_z \exp(-\beta \ \|p_z - \mu_z\|^2)$$

and the overall predicetd preference for a program $\mathbf{p}$ as:

$$\hat{r}(\mathbf{p}) = b + \sum_z B_z(p_z)$$

(i) How many parameters do you need to estimate in this case? Explain the role of each parameter.

[3]

Solution:
There is a total of $1 + 5 \times 3 = 16$ parameters: **[0.5]** - b is a constant parameter, the base rating when all values are far from the audience's preference. **[0.5]** - For each type of content z (5 types), we have three more parameters: **[0.5]** - μ_z is the quantity of content c that would lead to the strongest effect in terms of ratings. **[0.5]** - β_z is a scaling parameter influencing how broad or narrow is the bell curve, in other words how much a small departure from μ_c will impact the ratings. **[0.5]** - a_z is the relative importance of the type of content to the rating, similar to the linear case. **[0.5]**

(ii) What would be the most appropriate approach to fit this model to your data (Note: all of the functions above are differentiable, but B_c is clearly not linear)?
Explain how you would parametrise this problem (you are not asked to solve it!)

[3]

Solution:
Because the model is differentiable, gradient descent is likely to be the most efficient optimisation in this case **[1]**
The set of parameters is: $\theta = [b, a_m, \mu_m, \beta_m, \ldots, a_a, \mu_a, \beta_a]$ **[1]**
and the problem can be stated as:

$$\arg\min_\theta \sum_p \|\hat{r}(p) - r(p)\|_2^2$$

[1]

(c) Using this model $\hat{r}$, how would you use optimisation to find the best program, knowing that you want to run the radio from 6am to midnight daily, and need at least 1 hour of

advertisement per day to cover the radio running costs. How would you resolve this optimisation?

[2]

Solution:

The statement of the problem indicates that you need to use constrained optimisation **[0.5]**

$$\begin{aligned} \arg\max_{p} \quad & \hat{r}(p) \\ s.t. \quad & \sum \mathbf{p} - 18 = 0 \\ & 1 - \sum p_a \leq 0 \end{aligned}$$

[0.5] for the maximisation part. **[1]** for the constraints (0.5 each).

4. **(a)** Consider a relation Weather(Id, Time, Longitude, Latitude, Temperature, Humidity), where the primary key (Id) is a 116-byte string hash code, Time is 8-byte Datetime, the other fields are stored by 32-bit float. Assume that the relation has 30000 tuples, stored in a file on disk organised in 4096-byte blocks. Note that the database system adopts fixed-length records – i.e., each file record corresponds to one tuple of the relation and vice versa.

(i) Compute the blocking factor and the number of blocks required to store this relation. [2]

Solution:
(Bookwork for knowledge of the concepts, Problem solving otherwise)
Each record (tuple) has size: $116+8+4+4+4+4 = 140$ bytes. Thus, the blocking factor will be: $\lfloor \text{ block size } \rfloor = \lfloor \frac{4096}{\text{record size}} \rfloor = 29$ records per block. The file will then contain:

$$\left\lceil \frac{\text{num tuples}}{\text{blocking factor}} \right\rceil = \left\lceil \frac{30000}{29} \right\rceil = 1035 \text{ blocks.}$$

[1] for the computation of the blocking factor, **[1]** for the number of file blocks.

(ii) You are told that you will need to frequently add new records and you will not often read and fetch a record. Describe in detail the file organisation that you would expect to exhibit the best performance characteristics. Explain your answer by comparing the cost of reasonable alternatives.

[3]

Solution:
(Bookwork for knowledge of the concepts, Problem solving otherwise)
The relation can be stored in a number of ways: heap file, sequential file (sorted on one or more attributes), hash file (hashed on one or more attributes). Of these, heap files are expected to have the best performance in this relation since inserting a new record is efficient, for a cost of $O(1)$. **[1]**
With sequential file, all the records are ordered by an ordering field, e.g., name, and are kept sorted at all times, which suitable for queries requiring sequential processing. **[1]**
With hash file, reading and fetching a record is faster compared to other methods as the hash key is used to quickly read and retrieve the data from database. **[1]**

(b) Consider the following three relations:

- Student(Id, FirstName, LastName, DateOfBirth) where
  - the primary key (Id) is a 32-bit integer,
  - FirstName and LastName are both 96-byte strings, and
  - DateOfBirth is a 32-bit Integer.
- Course(Id, Description, Credits), or *C*, where:
  - Id, the primary key of this relation, is a 32-bit integer,
  - Description is a 195-byte string, and

  - Credits is an 8-bit integer.
- Transcript(StudentId, CourseId, Mark), or T , where:
  - StudentId is a foreign key to the primary key (Id) in the Student relation,
  - CourseId is a foreign key to the primary key (Id) in the Course relation above
  - Mark is a 8-byte double precision floating number, and
  - the primary key consists of the combination of StudentID and CourseID.

Assume these relations are also organised in 4096-byte blocks, and that:

- Relation Course (C) has r_C = 32 records and n_C = 2 blocks, organised in a heap file,
- Relation Transcript (T) has r_T = 51200 records and n_T = 200 blocks, organised in a sequential file, ordered by StudentID.
- Relation Student (S) has r_S = 2000 records and n_S =100 blocks, stored in a heap file and has a 4-level secondary index on *StudentId*.

Further assume that the memory of the database system can accommodate $n_B = 23$ blocks for processing and that the blocking factor for the join-results block is $bfr_{RS} = 10$ records per block.

Last, assume we execute the following equi-join query:

```
SELECT * FROM Transcript AS T, Student AS S, Course AS C
  WHERE T.StudentId = S.Id AND T.CourseId = Course.Id
```

As this is a 3-way join, assume that you need to join T with C first, with each block of intermediate results stored only in RAM (in one of the n_B blocks), then joined with S.

(i) Describe the join strategy that would be the most efficient in this case and estimate its total expected cost (in number of block accesses). Show your work.

[8]

Solution:
Join T with C first, then their result with S.

- First join-Intermediate $(I) = T \bowtie C)$:

$$js_{I=T\bowtie C} = 1/\max(NDV(CourseId, Transcript), NDV(Id, Course)) = 1/\max(32, 32) = 1/32$$

. **[1]**

$$jc_{T\bowtie C} = js_{T\bowtie C} \times |T| \times |C| = 1/32 \times 51200 \times 32 = 51200$$

. **[1]**

- This join can also be computed in two ways:
  (a) Scan C at the outer loop, do binary search on T for each value $(\text{ceil}(\log_2(n_T)) = 8)$:

With 51200 records for 2000 students, each StudentID will appear in 26 records in T on average;

with 16 bytes per record in T and 4096-byte blocks, all 26 records should be in one block (on average). The cost of this join is then: $n_C + r_C \times (\text{ceil}(\log_2(n_T))) = 2 + 32 \times 8 = 258$ block accesses. **[1]**

(b) Scan T at the outer loop, full scan on C for each value

(i.e., Naive Nested Loop Join): The cost of this join is then:

$n_T + n_C \times \lceil \frac{n_T}{n_B - 2} \rceil = 200 + 2 \times 10 = 220$ block accesses. **[1]**

Of the above strategies, the second is the most efficient, with a total cost of 220 block accesses.

Second join $(I \bowtie S)$:

- This join can also be computed in two ways:

  (a) We can store the intermediate join results of each block in memory, then for each record we use $S.Id$'S 4 -level index.

$js_{I \bowtie S} = 1/\max(NDV(\text{ StudentId,I}), NDV(Id, \text{ Student })) = 1/\max(2000, 2000) = 1/2000$

. **[1]**

$$jc_{I \bowtie S} = js_{I \bowtie S} \times |I| \times |S| = 1/2000 \times 51200 \times 2000 = 51200$$

. **[1]**

There is only one strategy at this point:

For each record produced in I , search for the matching record in S.

S has a 4 -level index, hence it will take $4 + 1 = 5$ block accesses to locate each value. The cost of this join is then: $r_I \times 5 + \frac{jc_{I \bowtie C}}{bfr_{RS}} = 51200 \times 5 + \frac{51200}{10} = 261120$. **[1]**

The total cost of this plan: $220 + 261120 = 261340$ block accesses **[1]**

(b) We can also write the intermediate join results back to the disk, then use Naive Nested Loop Join:
The cost of writing back to $n_{CT} = \frac{jc_{T \bowtie C}}{10} = \frac{51200}{10} = 5120$ [1 mark]
The cost of this join is then:
$n_S + n_{CT} \times \lceil \frac{n_S}{n_B - 2} \rceil = 100 + 5120 \times \lceil \frac{100}{23 - 2} \rceil = 25700$ block accesses. [1 mark]
The total cost of this plan: $220 + 5120 + 25700 + 5120 = 36160$ block accesses. And this is the best solution. [2 mark]
Marking notes:

- 0.5 mark for each correct formula for join T with C, e.g. $n_C + r_C \times \log_2(n_T)$ or $n_T + n_C \times \lceil \frac{n_T}{n_B - 2} \rceil$.
- 0.5 mark for each correct formula for join S, e.g. $n_S + n_{CT} \times \lceil \frac{n_S}{n_B - 2} \rceil$ or $r_I \times (level + 1)$.
- 0.5 mark for each join selectivity formula or join cardinality.
- For the Second join, either of two ways will obtain a full 4 marks.

(ii) Compare the Naive Nested Loop Join and the Index-based Nested-Loop Join. Which one is faster? Explain why.

[2]

Solution:
The Nested Loop Join searches for a row in the entire inner side of the table / index. The Indexed Nested Loop Join searches for a row in the inner side of the index and seeks the index's B-tree for the searched value(s) and then stops looking further. **[1]**
The Index-based Nested-Loop Join is much faster than the Naive Nested Loop Join, since for the outer relation, he Naive Nested Loop Join need scan the entire inner relation while with Index-based Nested-Loop Join, there is an index on the join column of one relation can make it the inner and exploit the index, avoiding linear search and hence it is faster. **[1]**

Mark table

Question	Points	Score
1	15	
2	15	
3	15	
4	15	
Total:	60	