— Solution notes —

COMPUTER SCIENCE TRIPOS Part IB – mock – Paper 6

3 Foundations of Data Science (DJW)

I wish to compare two systems, A and B, along several dimensions of comparison. For each system $i \in \{A, B\}$, and for each dimension $j \in \{1, 2, 3\}$, I have collected five outcome measurements x_{ijk}, $k \in \{1, \ldots, 5\}$. I want to know if system B has systematically larger measurements than A, across all dimensions.

(*a*) Consider the model

$$X_{ijk} = \alpha_j + \delta 1_{i=B} + \text{Normal}(0, \sigma^2).$$

(*i*) Write this as a linear model. Write out the predicted outcomes for each of the six categories of (i, j), and give an interpretation of the δ term. [4 marks]

(*ii*) Give pseudocode to fit this model. Your code should estimate all the unknown parameters, including σ. [3 marks]

(*iii*) Give pseudocode to find a 95% confidence interval for your estimated δ, using resampling. [4 marks]

(*b*) The model from part (*a*) assumes that the difference between the two systems is common across all dimensions of comparison. I now wish to test this hypothesis. Consider the general model

$$X_{ijk} = \alpha_j + \delta_j 1_{i=B} + \text{Normal}(0, \sigma^2)$$

and let the null hypothesis H_0 be that $\delta_1 = \delta_2 = \delta_3$. Consider the test statistic

$$t = \sum_{i,j,k} \Big[x_{ijk} - \big(\hat{\alpha}_j + \hat{\delta}_j 1_{i=B}\big) \Big]^2$$

where hats denote parameters fitted under H_0. We'd expect this to be larger if H_0 were not true.

(*i*) Explain how to find the distribution we'd expect to see for t, under H_0. Give pseudocode. [4 marks]

(*ii*) Explain what is meant by a one-sided test versus a two-sided test. Which should we use in this case? [3 marks]

(*iii*) Give pseudocode to compute the p-value of this test. [2 marks]

Answer: Part (a). For linear modelling, we're dealing with feacture vectors and response vectors, and it's tremendously useful to explicitly put the data into spreadsheet-style format so that it's clear what those vectors are. The question describes data from two systems, three dimensions, and five repeats; in a spreadsheet we'd store it like this:

$$\mathsf{mydata} = \begin{array}{cccc} \mathsf{sys} & \mathsf{dim} & \mathsf{rep} & \mathsf{x} \\ \hline A & 1 & 1 & x_{A11} \\ A & 1 & 2 & x_{A12} \\ \vdots & & & \\ A & 1 & 5 & x_{A15} \\ A & 2 & 1 & x_{A21} \\ \vdots & & & \\ B & 3 & 5 & x_{B35} \end{array}$$

There are $2 \times 3 \times 5 = 30$ rows in total, and it doesn't matter what order the rows go in. In the answer below, when we refer to feature vectors, we're referring to length-30 vectors from this spreadsheet.

Part $(a)(i)$. The model can be written as

$$\mathsf{x} \approx \alpha_1 1_{\mathsf{dim}=1} + \alpha_2 1_{\mathsf{dim}=2} + \alpha_3 1_{\mathsf{dim}=3} + \delta 1_{\mathsf{sys}=B}.$$

The predicted outcomes are as follows. They don't depend on the repeat number k, so I'm writing $\mathsf{rep} = *$ to indicate "repeat this row 5 times, one for each k".

sys	dim	rep	pred.
A	1	$*$	$\hat{\alpha}_1$
A	2	$*$	$\hat{\alpha}_2$
A	3	$*$	$\hat{\alpha}_3$
B	1	$*$	$\hat{\alpha}_1 + \hat{\delta}$
B	2	$*$	$\hat{\alpha}_2 + \hat{\delta}$
B	3	$*$	$\hat{\alpha}_3 + \hat{\delta}$

Or we could write out the same table differently:

	$\mathsf{dim} = 1$	$\mathsf{dim} = 2$	$\mathsf{dim} = 3$
$\mathsf{sys} = A$	α_1	α_2	α_3
$\mathsf{sys} = B$	$\alpha_1 + \delta$	$\alpha_2 + \delta$	$\alpha_3 + \delta$

The parameter δ is the increase in outcome for B compared to A, assumed to be common across all dimensions of comparison.

Part $(a)(ii)$. This is a straightforward linear model, so we can fit it with least squares estimation.(To be precise, this is a probability model with Gaussian noise, therefore the maximum likelihood estimators for feature coefficients can be found using least squares estimation. See section 2.4 of lecture notes.)

```
model = sklearn.linear_model.LinearRegression(fit_intercept=False)
F = numpy.column_stack([indicator(mydata.dim==j) for j in [1,2,3]] \
                       + [indicator(mydata.sys=='B')])
model.fit(F, mydata.x)
```

Coding niceties: (1) our linear model equation doesn't include a constant "1" feature, so we have to set fit_intercept=False; (2) in creating the matrix F we have to concatenate the α features with the δ feature, and you can concatenate Python lists with +, which is not to be confused with element-wise addition of numpy vectors also using +. In an exam, all that's needed is pseudocode, not proper Python. Your answer should refer to the fact that there's no constant feature, but you don't have to give precise code to assemble all the feature vectors—you could sketch out a diagram, for example.

The question reminds us that we also need the maximum likelihood estimator for σ. As we've seen several times in lectures (section 2.4 of lecture notes; example sheet 1 question 6; example 7.5) the maximum likelihood estimator for σ is

$$\hat{\sigma} = \sqrt{\frac{1}{30}\sum_{i,j,k}(x_{ijk} - \text{pred}_{ijk})^2} \quad \text{where} \quad \text{pred}_{ijk} = \hat{\alpha}_{\text{dim}} + \hat{\delta}_{\text{sys}=B}.$$

```
# Fitted parameters
α1_hat,α2_hat,α3_hat,δ_hat = model.coef_

# Get the predicted values for each row of mydata
pred = α1_hat*F[:,0] + α2_hat*F[:,1] + α3_hat*F[:,2] + δ_hat*F[:,3]

# MLE for σ
σ_hat = numpy.sqrt(numpy.sum((mydata.x - pred)**2) / len(mydata))
```

Or you may remember that sklearn has convenient syntax for making predictions:

```
pred = model.predict(F)
```

Part $(a)(iii)$. As usual, we generate many synthetic datasets, and compute the quantity of interest ($\hat{\delta}$) on each of them. The code below uses parametric resampling to generate synthetic datasets; we could have also used non-parametric. Remember that parametric resampling means "generate new data, but use maximum likelihood estimators for each of the unknown parameters." Here, pred already includes the maximum likelihood estimators for $\hat{\alpha}_1$, $\hat{\alpha}_2$, $\hat{\alpha}_3$, and $\hat{\delta}$.

```
def rx_star(): return numpy.random.normal(loc=pred, scale=σ_hat)

def δ_mle(x):
    model = sklearn.linear_model.LinearRegression(fit_intercept=False)
    model.fit(F, x)
    return model.coef_[-1]

δ_hat_samples = numpy.array([δ_mle(rx_star()) for _ in range(5000)])

lo,hi = numpy.quantile(δ_hat_samples, [.025, .975])
```

- This code reuses the matrix F from before—we're just synthesizing a new x column, leaving the feature columns unchanged, so we don't need to recreate F here.
- This code produces a two-sided 95% confidence interval, but given that the question says "I want to know if B has larger measurements than A" it would arguably be more useful to give a one-sided confidence interval with hi $= \infty$.

Part (b). In lectures, we saw a general strategy for devising hypothesis tests: (1) write out a general model, (2) express your null hypothesis as a restriction on the parameters. That's exactly how this

question has specified H_0. It's usually useful to write out H_0 in terms of the parameters that it actually has: it says

$$X_{ijk} = \alpha_j + \delta 1_{i=B} + \text{Normal}(0, \sigma^2).$$

In other words, H_0 denotes exactly the same model that we looked at in part (a). The test statistic is

$$t = \sum_{i,j,k} \left[x_{ijk} - \left(\hat{\alpha}_j + \hat{\delta} 1_{i=B} \right) \right]^2.$$

It's easier to refer to this equation, than to refer to the equation given in the question and mentally add "where all the δ_j are equal".

Part $(b)(i)$. Under H_0, the model we're considering is exactly the model from part (a), so the resampler from part $(a)(iii)$ works perfectly well here.

The procedure: create many resampled datasets using `rx_star()`, evaluate the test statistic `t` on each of them, then plot a histogram of the result.

```
def t(x):
    model = sklearn.linear_model.LinearRegression(fit_intercept=False)
    model.fit(F, x)
    pred = model.predict(F)
    return numpy.sum((x-pred)**2)

t_samples = numpy.array([t(rx_star()) for _ in range(5000)])

plt.hist(t_samples)
```

One point to emphasize: the test statistic has to be a statistic, i.e. a function purely of the data. It shouldn't include unknown parameters, and it shouldn't include parameters fitted from a different dataset. Think of parallel universes…we want to know what a parallel-universe data scientist would conclude from seeing their dataset, and they can't see our dataset. Therefore, when we compute t on a resampled dataset, we also have to compute the model predictions for that fitted dataset, not re-use the fitted parameters for the dataset we actually saw.

Part $(b)(ii)$.

- If we expect t to be large if H_0 were false, then we'd consider large positive values of t to be evidence against H_0. This is called a one-sided test.
- If we expect t to be small (or large negative) if H_0 were false, then we'd consider small (or large negative) values of t to be evidence against H_0. This is also called a one-sided test.
- If we expect t to be either large or small if H_0 were false, and either is possible, then we'd consider extreme values of t on both sides to be evidence against H_0. This is called a two-sided test.

In this case, the question says "If H_0 is false, we'd expect $[t]$ to be larger", therefore we should use a one-sided test of the first sort.

Part $(b)(iii)$. The p-value is the probability of seeing a value as extreme or more extreme than what we actually saw.

```
p = numpy.mean(t_samples >= t(mydata.x))
```
