BD in Healthcare Apps, Homework #4

1. Answer homework questions below
2. When ready, submit your answers online as instructed

# Answer the questions

**Question 1:** (1 pt)

One can always accurately predict patient wait time from the...

1. Wait time of the previous patient
2. Current time
3. Patient age
4. **None of the above**

**Question 2:** (1 pt)

One of the most basic ways to build predictive models is known as

1. **Linear regression**
2. Linear segmentation
3. Health regression
4. Pearson correlation

**Question 3:** (1 pt)

Adding more predictors to the predictive model will always produce significantly better prediction quality

1. **True**
2. False

**Question 4:** (1 pt)

Reducing machine learning model dimensionality is important in healthcare because

1. It is more resistant to manual errors in the data
2. **It can help exclude missing features**
3. It can help create interpretable models
4. It is not really important

# Solve data problems

Time for building more models! – and now, after learning how to find and engineer their features, we can really look into using the most efficient, interpretable predictions. Model Interpretability is particularly appreciated in healthcare, where we want to understand why certain thing happen. Selecting the best short models is one of the best approaches to interpretability.

**Getting the data:** Please go to this ”Data Challenge” web page,

<https://medicalanalytics.group/operational-data-challenge/>

and click on the “Data only” link there to download the data[[1]](#footnote-1).

In the sheet, use **F3** tab (dataset) ONLY: delete F1, F2 and F4, to load faster in Python. Also*, drop all variables with names prefixed* with x\_ (these are the original timestamps, they make very little sense as model features).

**Wait** will be our target variable we want to predict/model.

The last sheet in this file explains the features – most of them were *engineered*, and you should understand *how* by now. But there are tons of them, and we have to sort this out.

Fitting the model: We can explore different machine learning models, but let’s pick linear regression, since it takes little time to compute. So if Y is the Wait, and X – everything else, you can fit regression with this code:

from sklearn import linear\_model

model = linear\_model.LinearRegression()

model.fit(X, Y)

Ypred = model.predict(X) # use trained regression model to predict

r = Y-Ypred # compute prediction error (residual)

e = abs(r).mean() # compute model error

To help you make sure that you stay on the right track, I provide a few *Checkpoints* after each question. The numerical values provided in multiple choices can be rounded – always pick the closest to your answer.

**Question 5:** (4 pts)

Using sklearn snippet above, build a linear regression model to predict **Wait** from all other columns (excluding the “x\_” columns, as I mentioned). Once the model is built, use it to compute the predicted wait value Ypred, and residual r (the difference between the true and predicted wait).

Let’s define *model error* as the average of its absolute residual, as shown above: *abs(r).mean()*

Find this error value e, and write it down.

What was the model error value ?

1. **23.25**
2. 25.25
3. 27.25
4. 29.25
5. 31.25

*Checkpoint: the median of absolute residual values, abs(r).median(), should be around 19.1*

**Question 6:** (2 pts)

Using the code below , understand and run Python built-in feature selection algorithm: RFE (Recursive Feature Elimination[[2]](#footnote-2)). In this code, “model” is the same regression model as you declared and used in the previous code.

# Run Python feature selection

if True: # just in case I want to disable this part

print('\n>Python feature selection:')

from sklearn.feature\_selection import RFE

from itertools import compress

for nFeatures in range(1,4):

rfe = RFE(model, n\_features\_to\_select=nFeatures)

X\_rfe = rfe.fit\_transform(X,Y) #transforming data using RFE

#Fitting the data to model

model.fit(X\_rfe,Y)

#print(rfe.support\_)

#print(rfe.ranking\_)

cols = list(compress( X.columns, rfe.support\_))

model.fit(X[cols],Y)

e = abs(Y-model.predict(X[cols])).mean()

print(e, cols)

Write down its output. What was the best 3-feature model error value?

1. 33.25
2. 32.25
3. **31.25**
4. 30.25
5. 29.25

*Checkpoint: All three models should include CardiacCount*

**Question 7:** (6 pts)

Can we always rely on built-in code? Let’s implement our own simplest best model selection as a fast-forward “greedy” algorithm[[3]](#footnote-3). It should work like this:

* Find the best one-feature model (try all one-feature models, and select the one with the lowest error e). This is our best feature F1.
* Using F1 from the first step, try adding one more feature to it (from all features you have left), to find the best 2-feature model (F1, F2)
* Similarly, keep adding more features: F3, F4, F5 – to the features from the previous step

What is the *e=abs(r).mean()* value for the best 3-feature model?

1. 34.7
2. 32.7
3. 30.7
4. **28.7**
5. 26.7

*Checkpoint: this model should contain LineCount0*

**Question 8:** (3 pts)

Feature selection is often used to create shorter, faster models. Why do we need several dozen features, if we can achieve a significantly smaller error with a only few?   
Run your greedy algorithm to find best models with up to 15 features (adding them one by one, as explained).

Using your greedy algorithm, what is the smallest number of features to get the error value under 24?

1. 4
2. 6
3. 8
4. **10**
5. 12

*Checkpoint: model error for 14 features is around 23.548*

**Bonus (optional question)**

**Question 9:** (6 pts)

The problem with simple feature selectors (RFE, “greedy”) is that they are fast, but suboptimal; they also produce a single model only. As a result, you can miss the best model; you will also miss other nearly-best models which can help interpret the outcome.

So let’s try another extreme: “brute force”. Consider *all possible combinations* of three features (F1,F2, F3). What is the best *abs(r).mean()* you can achieve now?

Clearly, you will have to use three nested loops for F1, F2 and F3. I strongly recommend saving all resulting models into a list. To do so, record each model as a tuple (error, F1, F2, F3), and keep appending those to the list of computed models. At the end, sort the list by the first tuple value[[4]](#footnote-4), to see which model won, and which came close.

Note, that this code will take a few minutes to run. So try it with 2-feature models first to make sure it runs well, and then add F3.

What is the *e=abs(r).mean()* value for the best 3-feature model?

1. 22
2. 24
3. 26
4. 28
5. 30

*Checkpoint:*

Note: You should see that this time we achieved a lower error compared to the 2- or 3- feature models from above. Moreover, you can see many models producing nearly optimal results. When you do data-science, always keep this in mind: the truth can be very multi-faceted, and one can learn a lot from these outcomes.

1. Please do NOT download the code – it is in Matlab, and was written for a different purpose. [↑](#footnote-ref-1)
2. <https://scikit-learn.org/stable/modules/feature_selection.html#recursive-feature-elimination> [↑](#footnote-ref-2)
3. “Greedy” algorithm always picks the best possible value at its current step. <https://en.wikipedia.org/wiki/Greedy_algorithm>. Note that RFE is greedy as well. [↑](#footnote-ref-3)
4. <https://stackoverflow.com/questions/10695139/sort-a-list-of-tuples-by-2nd-item-integer-value> [↑](#footnote-ref-4)