Databases and Data Mining in Astronomy

Olivier Aartsen
s1397478

Leiden University

Report on Exam Assignment

January 26, 2018

Problem 1

Problem 1a)

I have made the following structure for my database: every fits file with the same filter has been unified into a new table called exactly that filter, for example the fits files that contain the H-filter (*Field-1-H.fits*, *Field-2-H.fits* and *Field-3-H.fits*) are unified into one table called *H*. This is done so that the *StarID* remains unique in each table. Now for the Ks-filter this is more complicated since there are multiple exposures of one filter in a field, so I unified these into new tables based upon the order in which they were taken. So for example the fits files *Field-1-Ks-E002.fits* and *Field-3-Ks-E002.fits* have been unified into a new table called *Ks2*. Furthermore I have added the columns *ExpID* and *Field* to the new data-tables, the *ExpID* stands for exposure index and is a unique index for each exposure (so since we have 18 fits files there are 18 unique exposures and therefore *ExpID* ranges from 1 to 18). *Field* is just the field in which it was taken. Finally the *Info* table contains the information about each exposure where the *ID* in this table relates directly to the *ExpID* in the data-tables. I have also added one column; *Tablename* containing an easy to understand description of the exposure: for example *F1Z* stands for Field 1 and *Z* stands for the Z-filter (so it is the exposure taken in field 1 in the Z-filter). Now the final example is that of *F1KsE002* which has the same structure as explained for *F1Z* (so Field 1 and Ks-filter) but now it also has *E002* which means the second image taken in field 1 in the Ks-filter.

So why have I chosen this lay-out? Well for the following reasons:
- It retains the uniqueness of the *StarID* so that we can still use this as a primary key between data-tables.
- By unifying the original data-tables most of the queries become a lot easier, for example R2 becomes much easier since we only have to subtract values from two tables now (R3 becomes easier for the same reason). Furthermore all the queries can be done solely in SQL (for this database, later we'll see why for larger databases some queries become difficult to implement solely in SQL but this would be difficult for most if not all database layouts).
- The *Tablename* that was added to the *Info* table is a handy way of identifying what field and filter (and temporal order for the Ks-filter) we're looking at.
- Finally the added column *ExpID* to the data-tables makes it so that it's possible to relate every object with more information about the field in which it was taken (from the *Info* table). And the added column *Field* makes it so that we can easily select parts of the tables for a certain field without having to join the *Info* table to the data-table. It is a bit of redundant data, but it makes many query jobs simpler and so I think it's worth it.

So now for the formal schema's of the tables that I made, first the *Info* table:
ID INT, FieldID INT, Filename varchar(100), Filter varchar(2), MJD DOUBLE, Airmass DOUBLE, Exptime FLOAT, Tablename varchar(20), UNIQUE (ID), UNIQUE (Tablename), PRIMARY KEY (Tablename)

Now for the rest of the data-tables the schema is as follows:
RunningID INT, X DOUBLE, Y DOUBLE, Flux1 DOUBLE, dFlux1 DOUBLE, Flux2 DOUBLE, dFlux2 DOUBLE, Flux3 DOUBLE, dFlux3 DOUBLE, Ra DOUBLE, Dec DOUBLE, Class INT, Mag1 DOUBLE, dMag1 DOUBLE, Mag2 DOUBLE, dMag2 DOUBLE, Mag3 DOUBLE, dMag3 DOUBLE, StarID INT, ExpID INT, Field INT, UNIQUE(StarID), PRIMARY KEY(StarID), FOREIGN KEY(ExpID) REFERENCES Info(ID)

The python files that make and change the database including the tables are: *change_info.py* and *make_database.py*. The whole database is constructed automatically by running these two programs on the data and can therefore also be run on a new batch of files (provided that the same structure and names are used for this new batch). However before running *make_database.py* note that if you have an existing file called *prob1.db* in the directory in which you're running the program, it will be deleted (see the program itself for more information). Before we go any further let me show you the (recommended) way in which the programs should be executed (for this whole problem):
1. *change_info.py*
2. *make_database.py*

3. *query_database.sql*
4. *dump_tables.sql*
5. *plot_queries.py*
6. *simulation.py*

Also these python and sql files should all be run in the directory where all the data is present (which is the directory Tables for me). The python files are easily run in the terminal by: *python program.py* and the sql files can be run by first entering *sqlite3* in the terminal and then performing the following command: *.read program.sql*

Problem 1b

The following is my output (in the command shell) for the queries (obtained by reading *query_database.sql* in sqlite3):

```
R1:
So below is a list of the amount of stars with S/N > 5 in each image taken between MJD=56800 and MJD=
ExpID|nStars|Image
1|9862|F1Z
2|9985|F1J
3|9990|F1H
6|9992|F1KsE003
7|9967|F1Y
8|9968|F2Z
9|9985|F2J
10|9985|F2H
12|9985|F2Y
13|9930|F3Z
14|9983|F3J
15|9987|F3H
18|9977|F3Y

R2:
Here are the objects that have J-H > 1.5 (I limit the output to 5 to avoid filling the terminal):
StarID|JH
0|2.45
1|2.19
2|1.95
4|1.57
5|2.35

R3:
Here are the objects in field 1 where any of the measurements of the flux in the Ks-filter is more th
StarID|AverageFlux|Ks1Flux|Ks1Fluxerr|Ks2Flux|Ks2Fluxerr|Ks3Flux|Ks3Fluxerr
0|9055.49|7069.91|182.45|4714.6|144.59|15381.97|179.2
1|8808.17|6769.28|182.1|5876.42|146.33|13778.82|178.48
2|8447.66|5915.77|182.06|5616.17|145.08|13811.03|178.43
4|10683.84|7280.18|178.5|7631.56|176.5|17139.77|182.53
5|12822.83|9976.3|183.77|7125.33|147.0|21366.84|182.21

For field 2 this query does not make any sense since it only has one exposure in the Ks-filter. There

For field 3 there are 2 measurements of the flux in the Ks-filter, therefore we can perform this quer
StarID|AverageFlux|Ks1Flux|Ks1Fluxerr|Ks2Flux|Ks2Fluxerr
300022|139804.18|172116.91|276.82|107491.45|202.36
300054|138634.46|144592.17|250.51|132676.75|232.65
300057|92442.56|118057.67|239.83|66827.45|198.02
300097|73679.58|67054.49|213.26|80304.66|207.84
300099|48542.11|43933.43|202.56|53150.79|197.24
```

```
R4:
The following catalogues exist for field 1:
Tablename
F1Z
F1J
F1H
F1KsE002
F1KsE001
F1KsE003
F1Y
The following catalogues exist for field 2:
Tablename
F2Z
F2J
F2H
F2KsE001
F2Y
The following catalogues exist for field 3:
Tablename
F3Z
F3J
F3H
F3KsE002
F3KsE001
F3Y

R5:
The following stars have S/N > 30 for all filters in Field 1 (limited to 5 outputs):
StarID|MagY|SNY|MagZ|SNZ|MagJ|SNJ|MagH|SNH|MagKs1|SNKs1|MagKs2|SNKs2|MagKs3|SNKs3
8|15.8|70.49|16.4|51.47|16.8|34.75|15.8|35.28|14.0|76.03|14.9|32.27|14.0|175.31
22|16.2|48.41|16.8|35.88|16.2|59.67|14.6|103.06|14.4|55.92|14.2|71.06|14.4|128.83
32|16.3|42.37|16.9|35.06|16.5|47.84|15.0|75.91|14.7|41.5|14.7|46.45|14.8|89.66
35|16.3|46.58|16.8|37.36|16.4|54.07|14.8|90.16|14.6|43.15|14.5|52.8|14.6|109.83
37|16.3|43.83|16.9|34.3|16.3|54.1|14.7|96.65|14.5|50.53|14.3|60.0|14.5|119.88

The following stars have S/N > 30 for all filters in Field 2 (since Ks2 and Ks3 do not exist in this
StarID|MagY|SNY|MagZ|SNZ|MagJ|SNJ|MagH|SNH|MagKs1|SNKs1|MagKs2|SNKs2|MagKs3|SNKs3
170000|16.4|43.66|16.9|35.57|16.5|47.65|15.0|71.97|14.8|36.8||||
170006|15.4|99.98|16.0|81.95|15.5|116.06|14.0|174.18|13.7|96.91||||
170007|16.0|61.07|16.5|51.9|16.2|64.6|14.7|93.15|14.4|49.27||||
170012|13.8|379.28|14.3|309.61|14.0|396.98|12.9|454.46|12.3|308.72||||
170014|14.5|234.15|15.0|179.5|14.5|261.38|13.0|405.73|12.7|233.59||||

The following stars have S/N > 30 for all filters in Field 3 (since Ks3 does not exist in this field
StarID|MagY|SNY|MagZ|SNZ|MagJ|SNJ|MagH|SNH|MagKs1|SNKs1|MagKs2|SNKs2|MagKs3|SNKs3
300005|14.7|191.32|15.3|146.33|14.7|228.62|13.3|324.78|12.8|197.65|12.8|254.43||
300006|15.9|67.81|16.3|59.06|16.0|75.76|14.6|104.63|14.4|50.39|14.4|64.6||
300010|14.6|205.5|15.2|159.5|14.6|246.33|13.1|363.77|12.7|223.92|12.7|257.03||
300011|15.8|74.19|16.3|59.86|15.8|87.46|14.3|132.41|13.9|78.03|13.9|89.48||
300015|15.8|71.33|16.1|74.07|16.1|63.2|15.0|66.7|14.9|32.31|14.9|36.09||
```

So for the query R1 almost every object in every filter and field (between MJD 56800 and 57300) has a signal to noise higher than 5, meaning that the requirements for putting the data of an object into the fits-files must have been strict. For the query R4 there is not much more to say than what the output tells you.

For the queries R2, R3 and R5 the output consists of more than 20 entries (if I wouldn't have limited them to 5), so now follow some graphical results. These plots were obtained by dumping the results of these queries to .csv files by *dump_database.sql* and then these csv files are loaded in and plotted by *plot_queries.py*. First for R2, you can see in figure 1 below that there exists a certain distribution for these stars centered around approximately J-H=1.5. However since this approximate mean is close to the hard limit of 1.5 it is possible that the true mean of this distribution lies on a lower J-H.

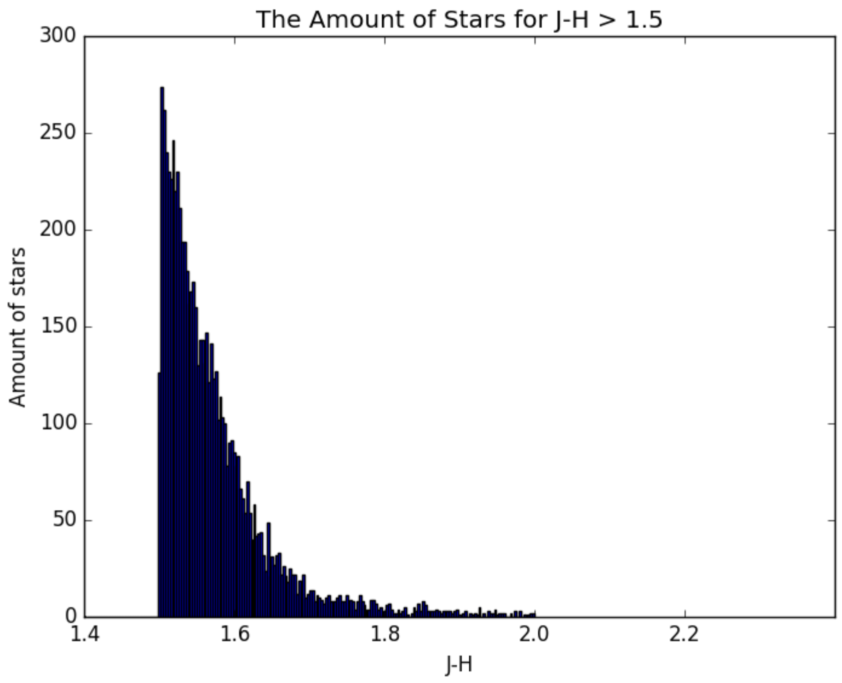

Figure 1: Here you can see the amount of stars with a certain J-H color (that is larger than 1.5).

Now for R3, you can see in figure 2 and 3 the maximum of the number of times the variation from the average flux is larger than the uncertainty in that flux for field 1 and 3 respectively. So this means that for a certain field the following is calculated:

$$max(\frac{F_{avg}-F_1}{\sigma_1}, \frac{F_{avg}-F_2}{\sigma_2}, ..., \frac{F_{avg}-F_n}{\sigma_n}) \tag{1}$$

where n is the amount of exposures in the Ks-filter in a certain field, F_{avg} is the average flux of all the exposures in the Ks-filter for every object, F_1 is the flux in the first exposure in the Ks-filter and σ_1 is the uncertainty in that flux. So in field 1 there are 3 exposures and so for every object this calculates the maximum of these three exposures in the amount of times that the uncertainty fits in the variation of the flux from the average flux. So basically we're looking at how large the temporal diversity in the flux is for the exposures in the Ks-filters. For field 2 there is only one measurement of the fluxes in the Ks-filter so there is nothing to compare this flux to. Now, figure 2 shows that for field 1 there seems to be a distribution about a mean of approximately 25, meaning that on average the measured flux for an object differs by about 25 times it's uncertainty to the average flux of that object in the Ks-filter, which is quite large. This seems to be because in the third exposure in the Ks-filter the fluxes are in general a lot larger than the fluxes measured in the first two exposures (this is not necessarily a problem for the third exposure in the Ks-filter in field 1, but it may just be because it got a longer integration time and therefore a higher flux, you could also see this in the data where the magnitude is indeed similar in the third exposure to those in the first two exposures since the magnitude is scaled appropriately).

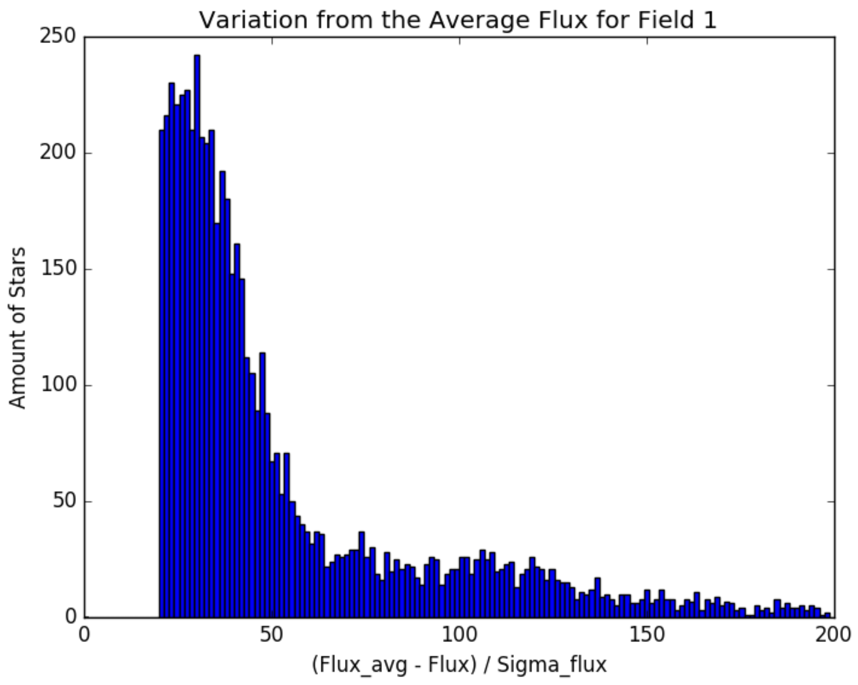

Figure 2: The maximum of the number of times the variation from the average flux is larger than the uncertainty in that flux for field 1.

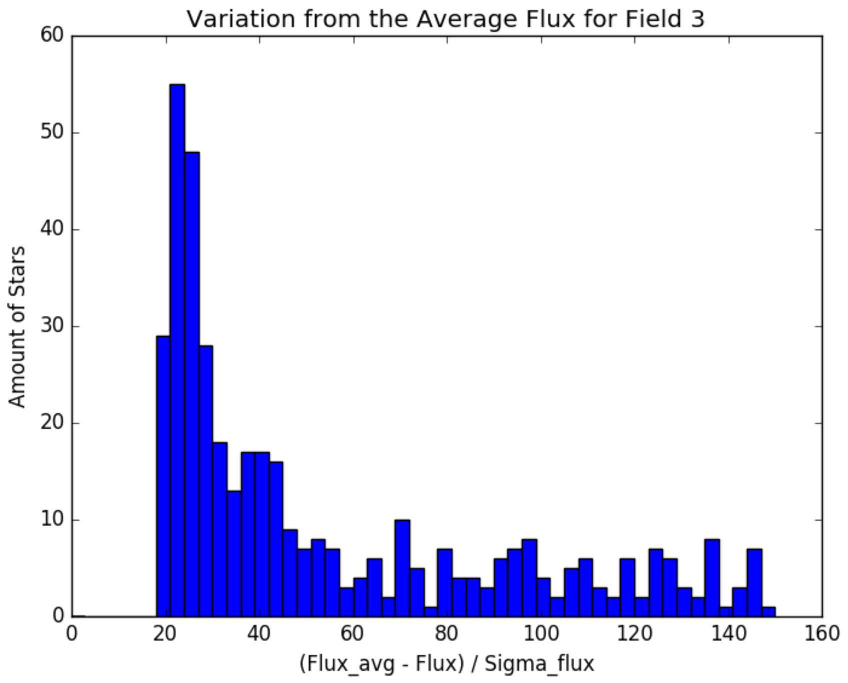

Figure 3: The maximum of the number of times the variation from the average flux is larger than the uncertainty in that flux for field 3.

In figure 3 you can see that for field 3 the amount of objects with a large temporal diversity is a lot smaller than in field 1 (which again could be explained that for some reason all the fluxes in the third exposure in field 1 are a lot larger than the fluxes in the first two exposures). It may look like the maximum of this distribution lies around 22, but we should not forget that this plot shows binned values. Therefore the bin that lies on 20 also could have had some values that are slightly smaller than 20, like 19.5, but due to the hard limit at 20 these values are not counted and therefore the bin seems lower in this plot than it actually may be had we plotted all the data.

Now in figure 4, 5 and 6 you can see the results for query R5 which show the signal to noise versus the magnitude for each star and for each filter with the additional constraint that in every filter the signal to noise was bigger than 30. Note that the query required the objects to be stars, which I did using the class data of each object (which is -1 for stars). In figure 4 can see for each filter that the data in the signal to noise ratio versus the magnitude follow a fairly nice relation. We expected this kind of relation because the lower the magnitude gets, the brighter the object gets and therefore the more probable it becomes that we get a higher signal to noise. The fact that these stars follow a different relation for different filters is not crazy either, since the flux and/or uncertainty for a star in different filters can in general be a lot lower or higher (due to atmospheric circumstances, integration time, etc...) and therefore the signal to noise for a certain magnitude can be different between filters. What is peculiar (or rather is not peculiar due to the results of query R3) is that the exposures in the Ks-filters give a different relation with respect to each other. The Ks1-exposure and Ks2-exposure seem close enough to each other, but the Ks3-exposure is way off. This again can be attributed to the higher fluxes (and therefore higher Signal to noise ratios for higher magnitudes).

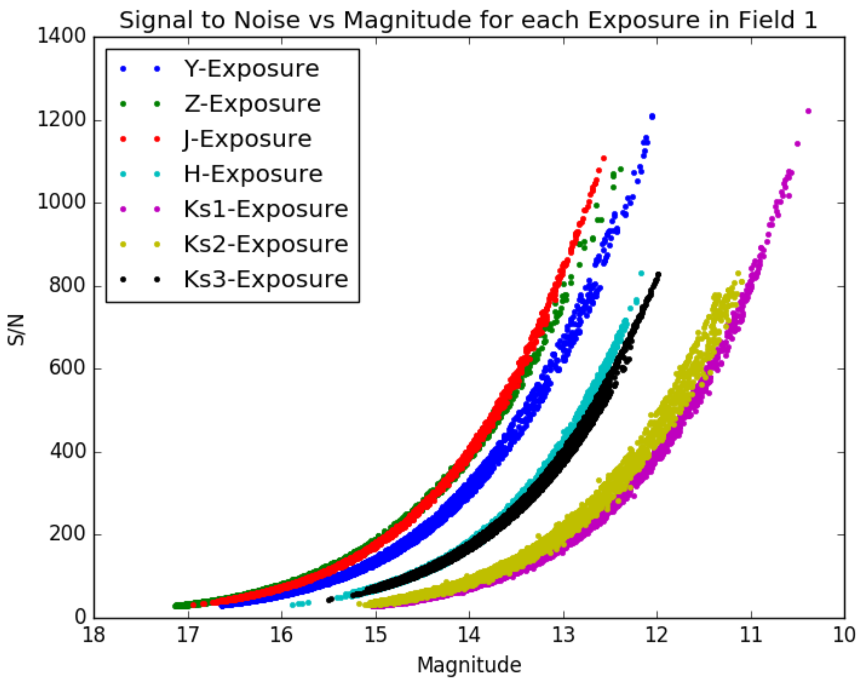

Figure 4: The signal to noise ratio vs the magnitude for each exposure in field 1.

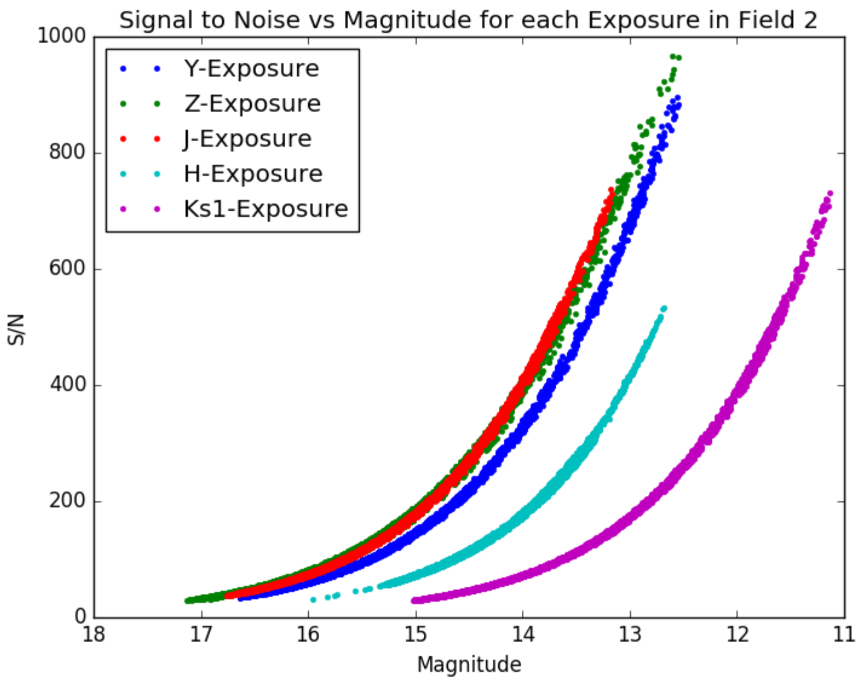

Figure 5: The signal to noise ratio vs the magnitude for each exposure in field 2.

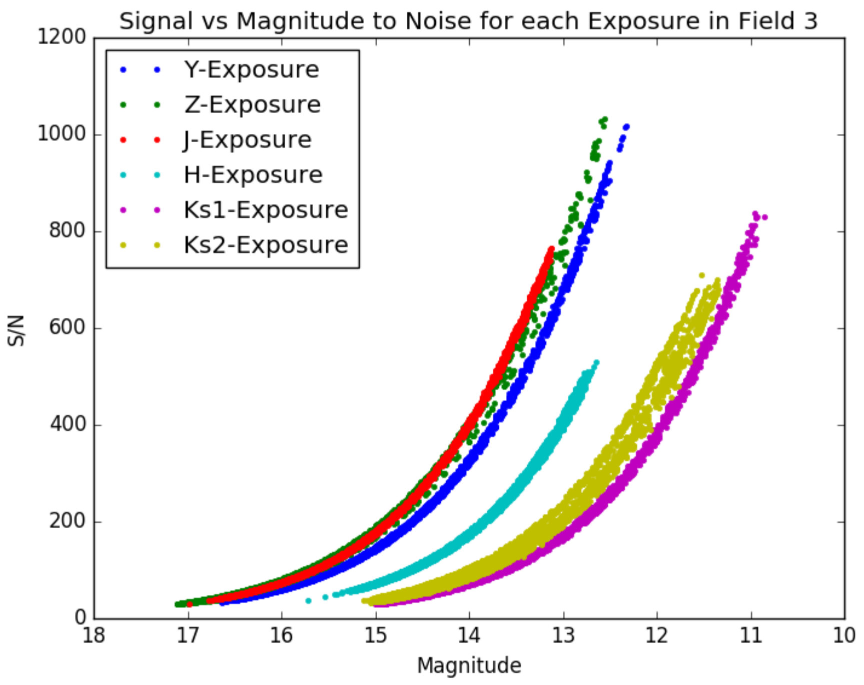

Figure 6: The signal to noise ratio vs the magnitude for each exposure in field 3.

For field 2 and 3 similar results are obtained, all though for these fields we of course only have respectively 1 and 2 exposure(s) in the Ks-filter.

Now some final words on the commands used for querying the database; these queries were done completely in sql, however for a more general and larger database with more exposures in the Ks-filter some of these queries could become problematic to do solely in sql (so without python assistance). For example for R3 we have to calculate certain values using data from each exposure in the Ks-filter, meaning that if we would have 50 exposures in this filter (and field) then we would have to make a command with 50 calculations in it for each filter (not even mentioning the 50 joins we would have to do). In sql this becomes a little problematic because for-loops are not easy to implement, while in python these would be much easier since for-loops are very easily implemented there. These problems are especially apparent for R3 and R5 but also the applicability of R1 in sql would suffer under more exposures in the Ks-filter. Now, I don't think this problem is avoidable by constructing a different database, since for R3 for example you still have to differentiate between the 50 Ks-filters no matter what lay-out your database has, which can only happen automatically under a for-loop. Therefore I think that my database structure works perfectly fine, even if many more exposures have to be accounted for, since then you would preferably do some jobs in python anyway.

Problem 1c

So first let's look at the distribution of stars in the Y-J, J-H plane that we already have (we have about 23700 stars). In figure 7 you can see this plot with the areas in the Y-J, J-H plane divided into some regions depending on their structure:

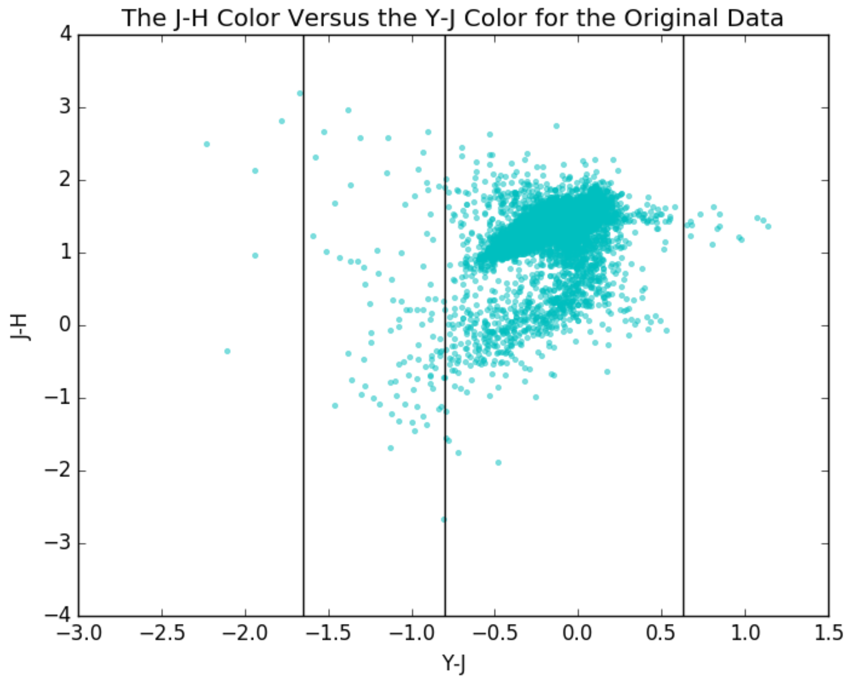

Figure 7: The original distribution of stars in the Y-J, J-H plane divided into sub-areas where some structure seems to be present.

So the distribution of stars in this plane seems to have structure, but the exact distribution is unknown and it seems that the distribution has some structure that looks like outliers (for example the most left sub-area seems to have some vertical structure, but these points look like outliers for computer programs). Therefore instead of trying to find this distribution it's a lot more efficient to study the distribution yourself and then simulate new points retaining these structures. This is exactly why we have the sub-areas in figure 7, to recognize the structure there and retain it later. From left to right (figure 7) the first sub-area has a couple of stars that are very diffuse in this plane in the vertical direction and fairly diffuse in the horizontal direction, so we create more variation in the vertical direction and a little less in the horizontal direction. The second sub-area is fairly diffuse in both the vertical and horizontal directions, but there also seems to be some structure where the plane is empty, therefore the variation in both the horizontal and vertical direction can't be too large. For the third sub-area the plane is very crowded as well as there seems to be structure, therefore the variation in both the vertical and horizontal direction will be made very small (but not negligible). Finally the fourth sub-are has some structure mostly in the horizontal direction and therefore we will retain this structure by varying more in the horizontal direction and less in the vertical direction. The way in which new data-points were simulated is: randomly choose a point from the original data, determine in which sub-area (from figure 7) this point is, accordingly add random noise in the horizontal direction and (if needed add a different random noise to the) vertical direction and then we have a new data-point. In this way 100000 new data-points were simulated which you can view in figure 8. Also the simulated data is written out to a csv file (by *simulation.py* to a new csv file called *sim_JH_YJ.csv*).

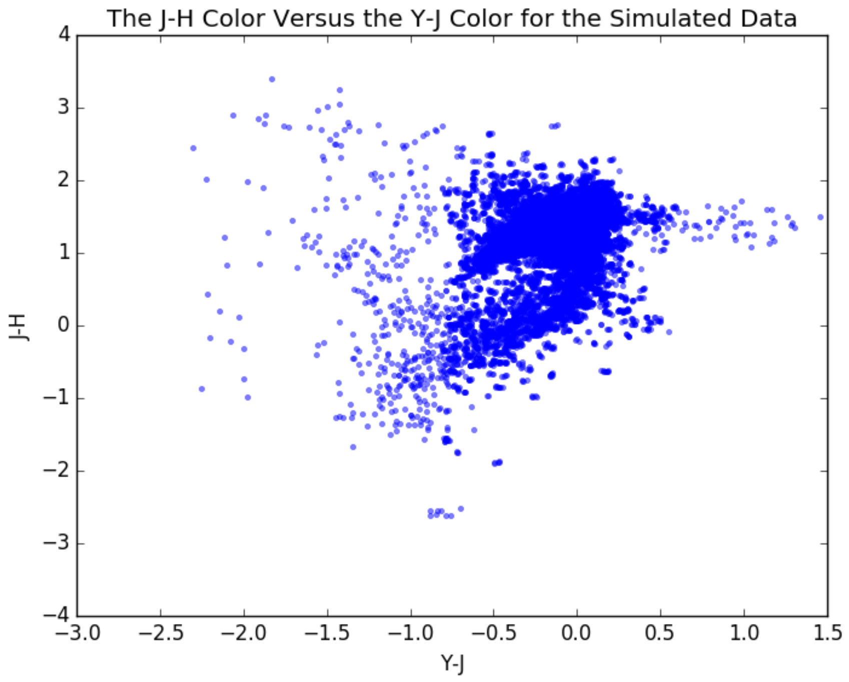

Figure 8: The simulated distribution of stars in the Y-J, J-H plane.

Looking at figure 8, the structures that were present in the original data seem to have been retained while creating these 100000 data-points. So the data is ready to be used by our colleague.

Problem 2

Problem 2a

Before I answer the question note that the programs should all be executed in the Tables directory, furthermore they could be run in any order (since there are no files created that another program uses) but the chronological order is as follows:
1. *regression.py*
2. *gen_error.py*
3. *nonlinear_estimator.py*

Now to answer the question: Feature extraction is the act of transforming a dataset to a smaller set of so called 'features'. If performed well the features will describe the data with satisfactory accuracy for your end goals. Often in a dataset some data is correlated with another part of your data and/or you're only interested in some part of the data and/or the dataset is so large that it would take too much time/power to perform your requested analysis on. In these cases you would prefer a smaller dataset with exactly those features that you want to investigate and that is exactly what feature extraction aims to make true.
For a classification/fitting task you want to make sure that your different parts of your data aren't correlated in a way that they describe the same feature. Also you want to make sure that you don't retain irrelevant parts of your data, since they just make the data more noisy. And finally you want to try and get the best results out of your data with the available computing power (and time), so if you remove too much of your data you might not be able to classify/fit the data with high accuracy at all (no matter how many times you try to improve your classification/fit), but if you remove too little from your data then the time it will take for your computer to run the classification/fit method with a good enough accuracy could become too large.

Problem 2b

First of all, the code that can be run for the regression is called *regression.py*. Now, to solve this problem let's first look at the plot of the correlation between the data that I have produced in figure 9:

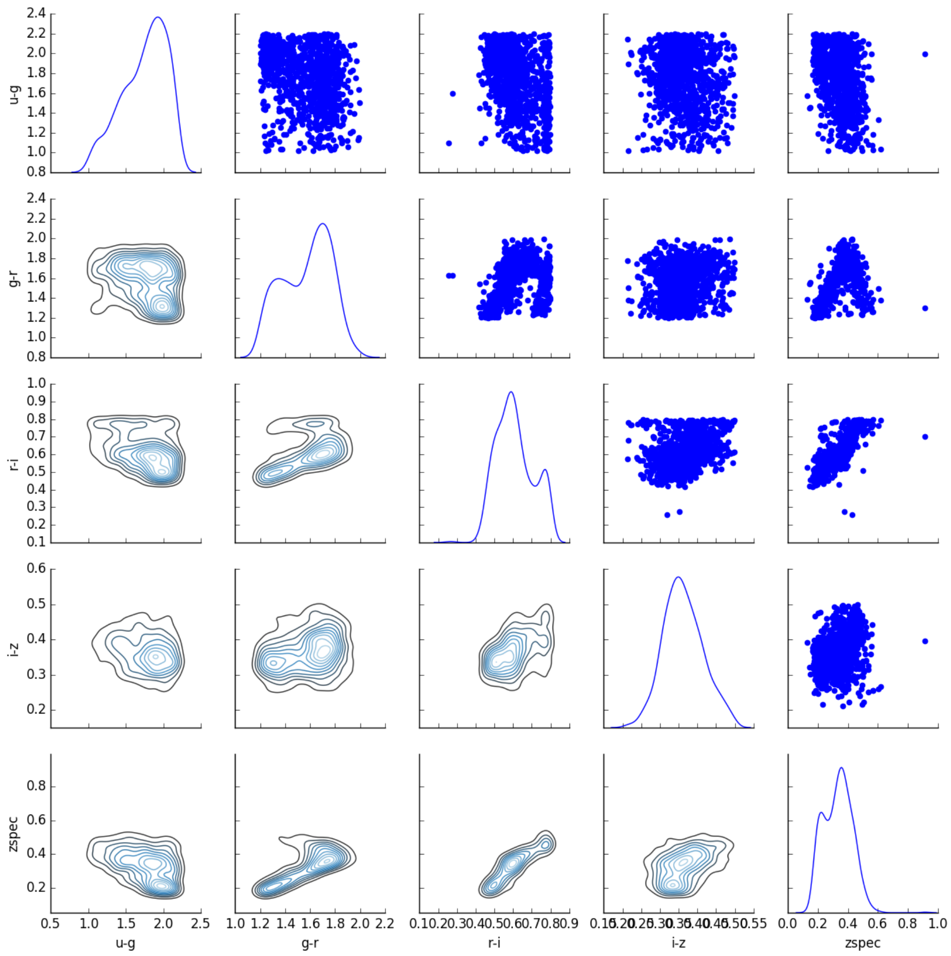

Figure 9: The correlation between the different colors and the spectral redshift z_{spec}.

So as you can see, the data don't seem to be correlated very much, but there still is some significant correlation. For example z_{spec} seems to have significant correlations with all the colors. Some of the colors don't seem to be correlated much with each other but some of them do show significant correlations. For these reasons Ridge regression seems to be the most appropriate regression type to perform on this data. This is because Ridge regression will not put any of the coefficients to zero but Lasso might which is not what we want. Furthermore since there seems to be some correlation between the colors we don't want to use Linear Regression (since that would assume no or almost no correlation between the colors). Personally I don't think linear regression is that bad in this case, it just doesn't completely seem like the appropriate regression type to use. So we do assume that:

$$z_{spec} = \theta_0 + \theta_1 * (u - g) + \theta_2 * (g - r) + \theta_3 * (r - i) + \theta_4 * (i - z) \tag{2}$$

After performing the regression, the program *regression.py* puts out the best-fit model which is:

$$z_{spec} = -0.214 - 0.027 * (u - g) + 0.115 * (g - r) + 0.666 * (r - i) + 0.011 * (i - z) \tag{3}$$

We have tried to get an as low as possible training error, which is: 0.0151 (which is just above the 0.014 that Jarle said we could get, so I'm fairly close). Finally the program puts out a plot of the residuals (on the y-axis) versus the redshift (on the x-axis), see figure 10:

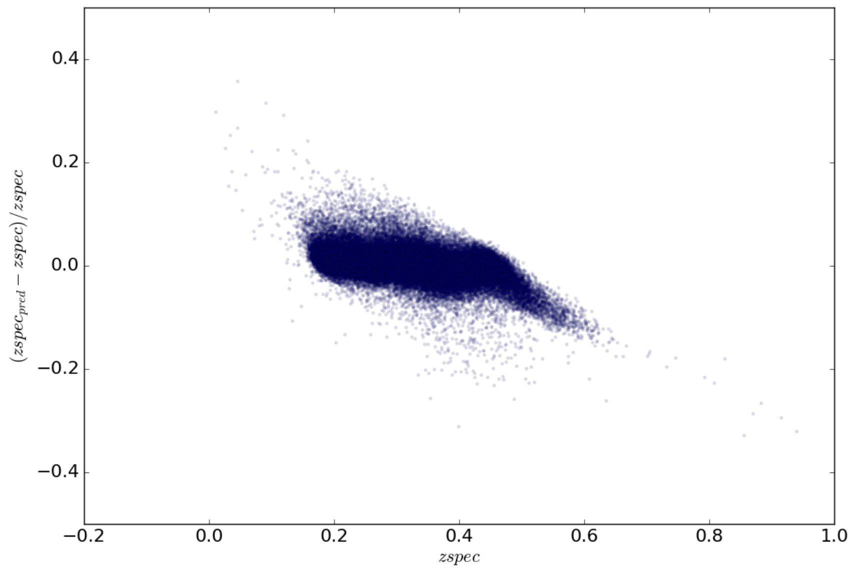

Figure 10: The residuals versus the redshift for the lowest training error regression.

Problem 2c

So what we have done in 2b) is trying to minimize the training error, which is just the error for the particular training dataset. This means that it is especially designed to get an error as low as possible for this particular dataset, not for a different dataset. Therefore by putting a different dataset with this model we could get a much worse error, so the generalization error is not estimated reliably by the estimated error on the training dataset. We achieved the low training error by putting the regularization strength in the *sklearn.linear_model.Ridge* function to 0.01 (very low). This means that the variance of the estimates can be fairly large, so indeed this regression wouldn't necessarily fit every dataset as nicely as it did this one.
Now in the program *gen_error.py* I calculate the residuals coming from the dataset *PhotoZFileB.vot* using the model we found in 2b, and from these residuals I calculate the generalization error which results in: 0.0153. So this result is actually very close to the training error, so apparently the two datasets have very similar correlations in the data.

Problem 2d

I tried to use the Random Forest method to implement a photo-z estimator, but unfortunately my computer wouldn't allow it. It would start the program, the computer would more or less freeze and after about ten minutes it would kill the process automatically without any error message (so probably it just took too much memory for the computer to handle). Unfortunately I do not have enough time left to look into this, all though the program (which might contain errors since I couldn't develop it further) is still there, so if you want you can try running it. Now I will still give a brief answer to question 2e) as for what I expect the advantages and disadvantages to be.

Problem 2e

So the advantages come down to a more precise prediction of new data when there are nonlinear correlations (as you could see in the generalization error becoming lower), so in general it just does a better job than the linear regression methods unless the correct model is completely linear. And the disadvantage is that it runs a lot slower (as I said in 2d it basically froze my computer), so if you don't have enough computing power or time you might want to reconsider. Also if you know that the correct model is linear (you just don't know exactly the coefficients) then you should use linear regression methods and get a model that is as accurate as possible.