DDM 2017 - Final problem set

Len Hartsuiker – s1498738

January 25, 2018

1 A database for a time-domain survey

a) Creation of a schema for the database

The first task of this final problem set is to create a database that can keep track of the location of the reduced images, the catalogues of detection in each image, the colours of the stars and the variability of the stars in the Ks band. There are five sample queries which are a test for the structure of the database. When these queries can be solved in a fairly easy and general way, it means that the database is suitable to meet the scientific needs of the team of the survey manager.
After investigating the options and considering which one was the best for the given sample queries, I decided to make a different table for every filter. When using this structure, queries selecting on a magnitude, flux or combination of magnitudes or fluxes of a certain colour become very straightforward. Since three of the five queries need this kind of selection (and the query needed for the Euclid space mission as well), this structure seems to be a sensible choice. I made another table for the 'file_ info_ for_ problem.csv' table, which I will refer to as 'mastertable' from now on. In the filter tables I also added an 'ImageID' column, corresponding to the 'ID' from the mastertable, so it is possible to link a filter table to the corresponding image in the mastertable. Due to this addition it becomes possible to extract part of the filtertables, when only certain images are required (for example images for a given field). Since the column 'Filename' in the mastertable didn't contain the right filenames of the filter tables, I updated this column. This makes it a lot easier to assign the right ImageID to an observed object in the filter tables in a general way.
Some queries extract the magnitude or the flux of certain objects. In the database there are three definitions of each of these quantities, corresponding to three different apertures. In these queries I will always use the first aperture, which has a diameter of one arcsecond. Furthermore, for some fields there are different images in the Ks filter. In the first and third sample query, I will extract the results of all Ks images, to give a complete result of the data. For the fifth query I will only use the image that was taken first, since the result of that query just wants to extract high S/N objects in all filters and therefore we only need one of the images.

b) Devision of the SQL queries

The SQL queries used to solve the sample queries can be found in the python code. The queries can be done fully in SQL (when you implement the desired field on the places where a given field is requested), but I thought implementing them in Python would be a nice way to have an overview of the devised queries and to be able to change between fields easily.

R1: Find all images observed between MJD=56800 and MJD=57300 and give me the number of stars detected with S/N > 5 in each image. For the first query I glued the rows that fulfilled the restrictions from the different filter tables together using the UNION statement. Then, by grouping on the ImageID I get the results shown in Table 1. Since every image contains 10,000 stars, almost every object fulfils the signal-to-noise requirement.

R2: Find the objects that have J-H > 1.5. Queries like this are particularly easy with the chosen structure for the database. Since the result consists of more than 20 entries, the result is visualized in the histogram in Fig. 1.

Table 1: Results query R1

ImageID	Number of Stars with S/N > 5
1	9862
2	9985
3	9990
6	9992
7	9967
8	9968
9	9985
10	9985
12	9985
13	9930
14	9983
15	9987
18	9977

R3: Find the objects where Ks differs by more than 20 times the flux uncertainty from the mean flux. For this query I used all the images in the Ks filter (even those that are done on the same StarID object). By doing this, one can see whether an object always has a 'weird' value or that it was a fluctuation only during a particular image. It should be noted that, despite the fact that all images have the same exposure time, the flux averages differs enormously between some of the images, ranging between values of 13883 and 24905. Therefore I compare the fluxes of the objects with the average flux of their image rather than the average flux of all images combined. The results are visually presented in the histogram of Fig. 2.

R4: Find all catalogues that exist for a given field. For the fourth query we only need the mastertable, so this query is rather straightforward. The results are shown in Table 2.

Table 2: Results query R4

Field	ImageID's
1	1,2,3,4,5,6,7
2	8,9,10,11,12
3	13,14,15,16,17,18

R5: For a given field I would like to retrieve the Y, Z, J, H and Ks magnitudes for all stars with S/ N > 30 in Y, Z, J, H and Ks. For the fifth and last sample query I joined all filter tables, since we are only interested in the objects that fulfil the signal-to-noise requirement on all filters simultaneously. Since some object are measured more than once in the Ks filter, I only selected the flux values that were measured first. I did this since we are now interested in objects that have a high signal-to-noise in every filter, so one image in Ks suffices. The results of the first field are shown as a multivariate kernel density plot in Fig. 3, the results of the second field in Fig. 4 and the results of the third field in Fig. 5.

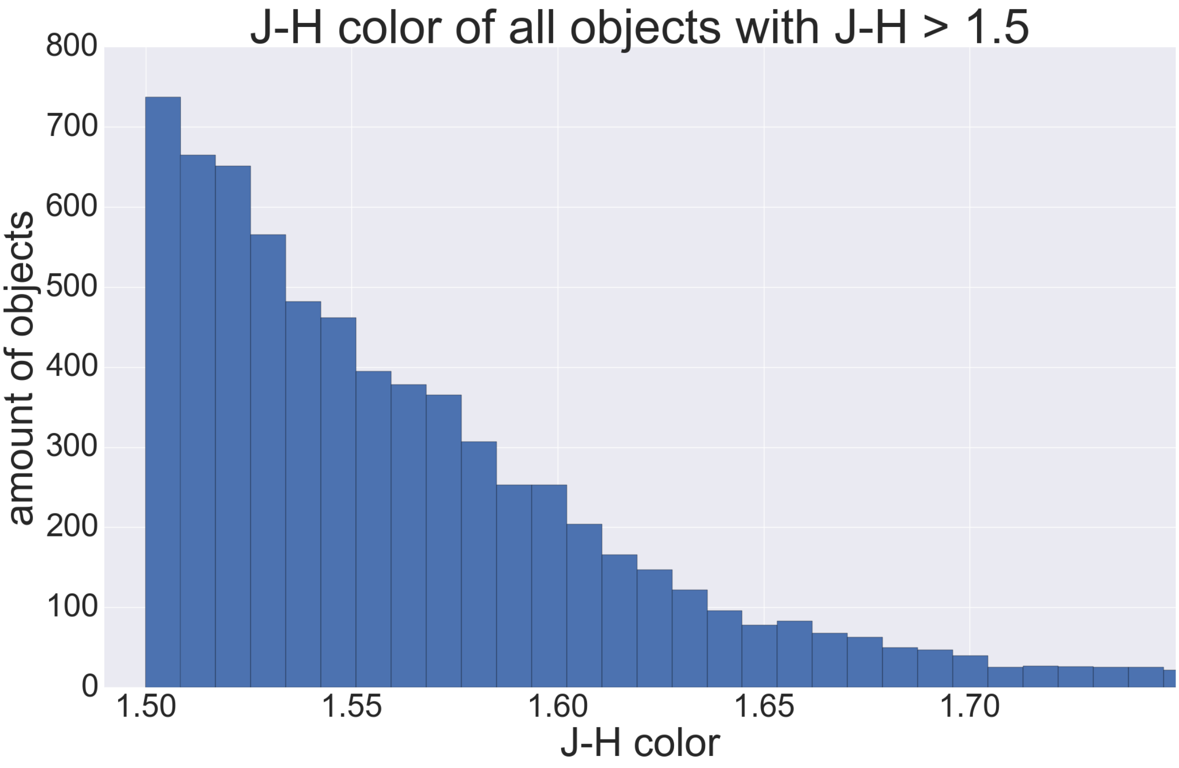

Figure 1: Histogram of the J-H color for all objects fulfilling the restriction. It should be noted that the histogram is capped at J-H = 1.75 for visibility. Beyond 1.75 the tail of the histogram starts to shrink until is vanishes completely at J-H = 3.2.

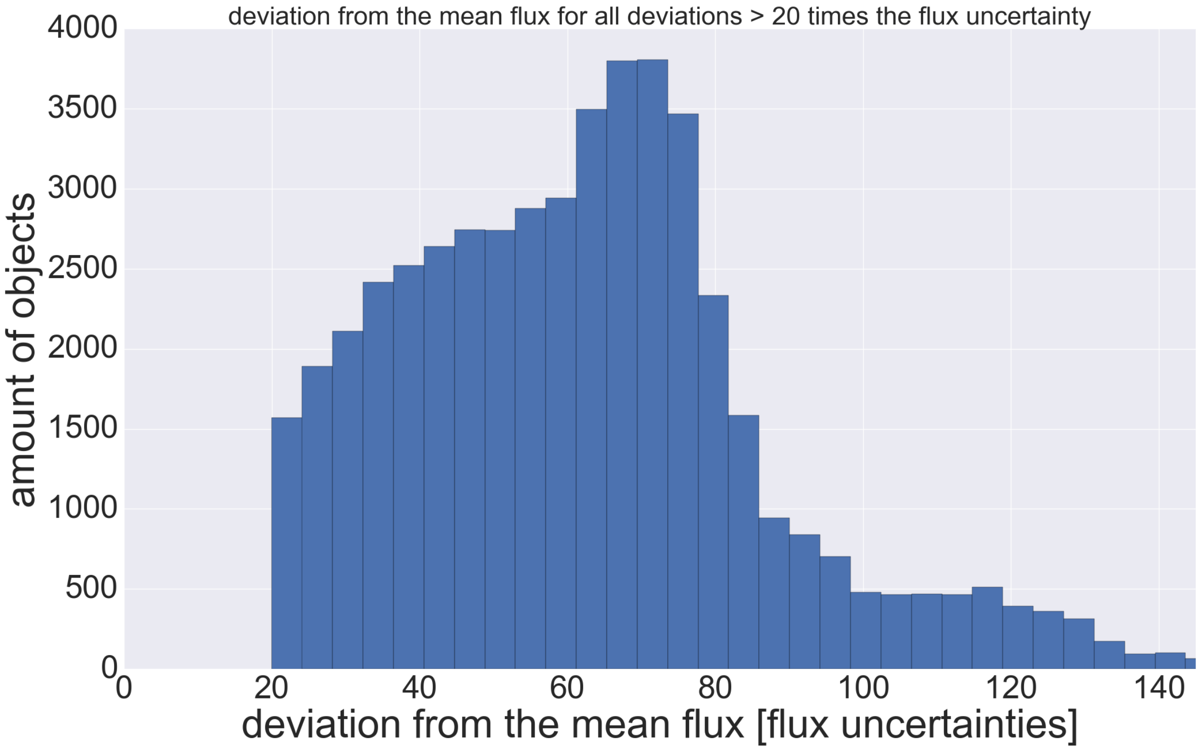

Figure 2: Histogram of the deviation from the mean flux in terms of flux uncertainties for all objects observed in the Ks filter. The histogram is capped at 150 for visibility. Beyond 150 the tail of the histograms shrinks until it vanishes at 800. Beyond 800 there are some sporadic distributed values up to 1172.

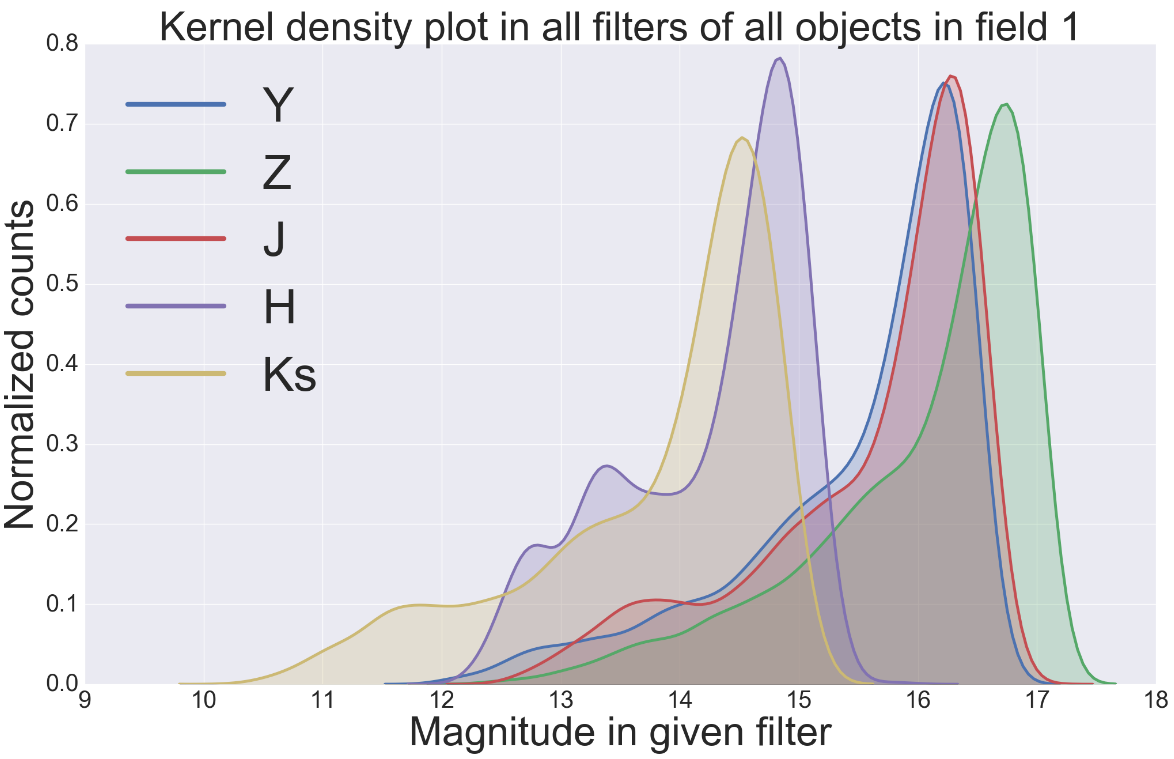

Figure 3: Kernel density plot of all magnitudes in field 1.

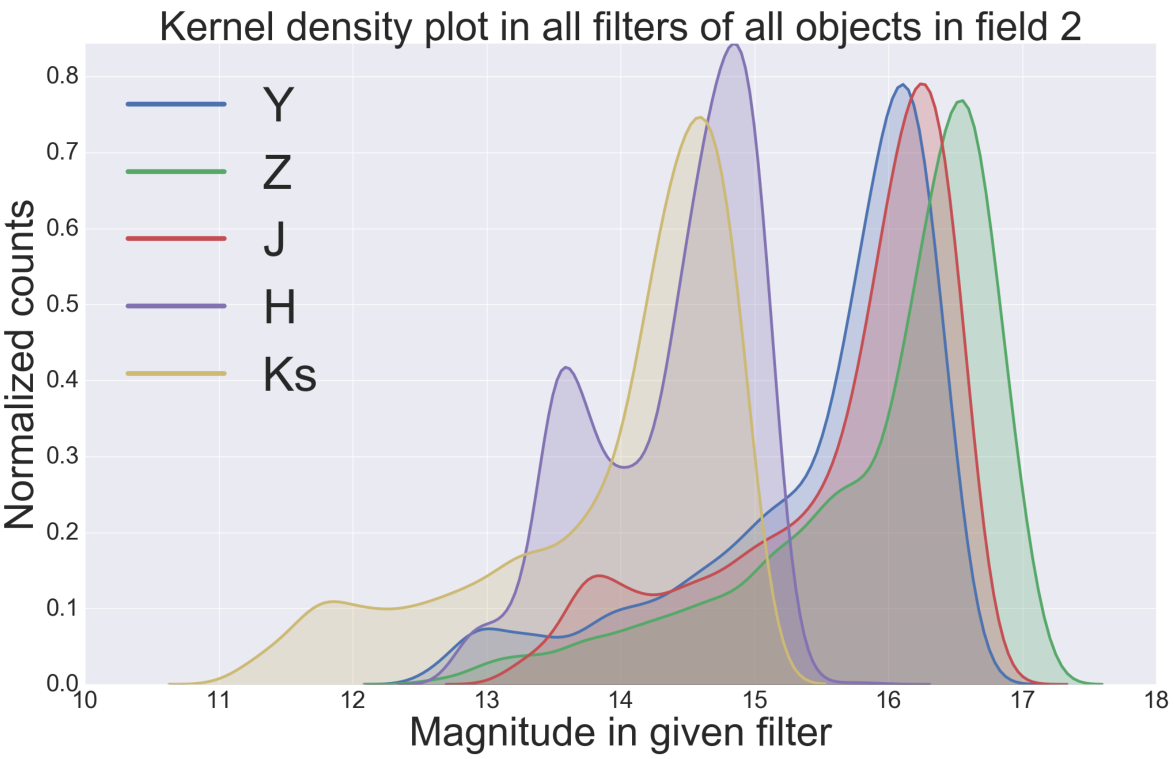

Figure 4: Kernel density plot of all magnitudes in field 2.

c) Simulation sample for Euclid

As a start I first query all Y-J and J-H combinations out of the database. I reject all values where either of these colours is NULL, since NULL values will give errors later on. To estimate the probability density function of the stars in Y-J, J-H space I use a kernel density estimation. I have chosen for this non-parametric estimator (the bandwidth is a free parameter) since I am not interested in the science behind the selection of the sample. To determine which bandwidth will perform best, I take a sample of 2000 stars from the database and perform a 10-fold cross-validation on these stars. Then I loop over a range of bandwidths to determine which bandwidth maximizes the log-likelihood. I use this bandwidth (which happens to be 0.059) to perform a kernel density estimation on the Y-J and J-H combinations of all stars from the database. From this kernel density I can now create a new sample of 100,000 stars to use as simulation sample for Euclid. The result as a scatter plot in the Y-J, J-H plane is shown in Fig. 6. The result as a 2D distribution function is shown in Fig. 7. It should be noted that the values on the axes are quite different. The reason for this is that Fig. 7 shows where most points are distributed and the majority of data points is distributed close to each other. On the other side, Fig. 6 shows all points, including the outliers, which are sometimes well separated from the majority of the data points.

2 Photometric redshifts of galaxies

The goal of this exercise is to predict the redshifts of galaxies using photometry. Since the requirements of large future cosmological surveys are that

$$\left\langle \left| \frac{z_{phot} - z_{spec}}{1 + z_{spec}} \right| \right\rangle < 0.01,$$

or even smaller, we will use

$$E(\theta) = \text{median} \left(\left| \frac{z_{spec} - f(\theta)}{1 + z_{spec}} \right| \right)$$

as the measure of the discrepancy between photometric and spectroscopic redshift. In this expression, f is a function of our available photometric parameters, namely the r-band magnitude of the galaxy and the u-g, g-r, r-i and i-z colour of the galaxy.

a) Feature extraction

After having extracted the data from the database, the second step in the classification process is feature extraction. Feature extraction is the transforming of the data from a space with a certain amount of dimensions to a space with fewer dimensions. In this process the redundant variables are discarded, leading to a simpler model which often leads to better interpretations of the model and less computational time. The caveat of feature extraction is that you can throw away information when reducing the dimensionality. Therefore you should always consider whether the simplicity gained with feature extraction is worth the loss of information. Based on the size of the database one could choose to use feature extraction if this would reduce the computational time significantly. Another consideration is whether one want to understand the model (in which case feature extraction is recommended) or that it is more like a black box kind of model (in which case feature extraction is less important).

In the case of our database feature extraction could be implemented since we have three related quantities that all depend on the r-band magnitude. These relations can be seen in the scatter plot matrix of Fig. 8, where 500 datapoints are plotted.

However, this database is not so large that it will cost a lot of computational time. Furthermore, the goal of the exercise is estimating the discrepancy between photometric and spectroscopic redshifts rather than understanding how this estimation is done. Therefore I will not implement any feature extraction during the classification process as I would like to preserve as much information as possible.

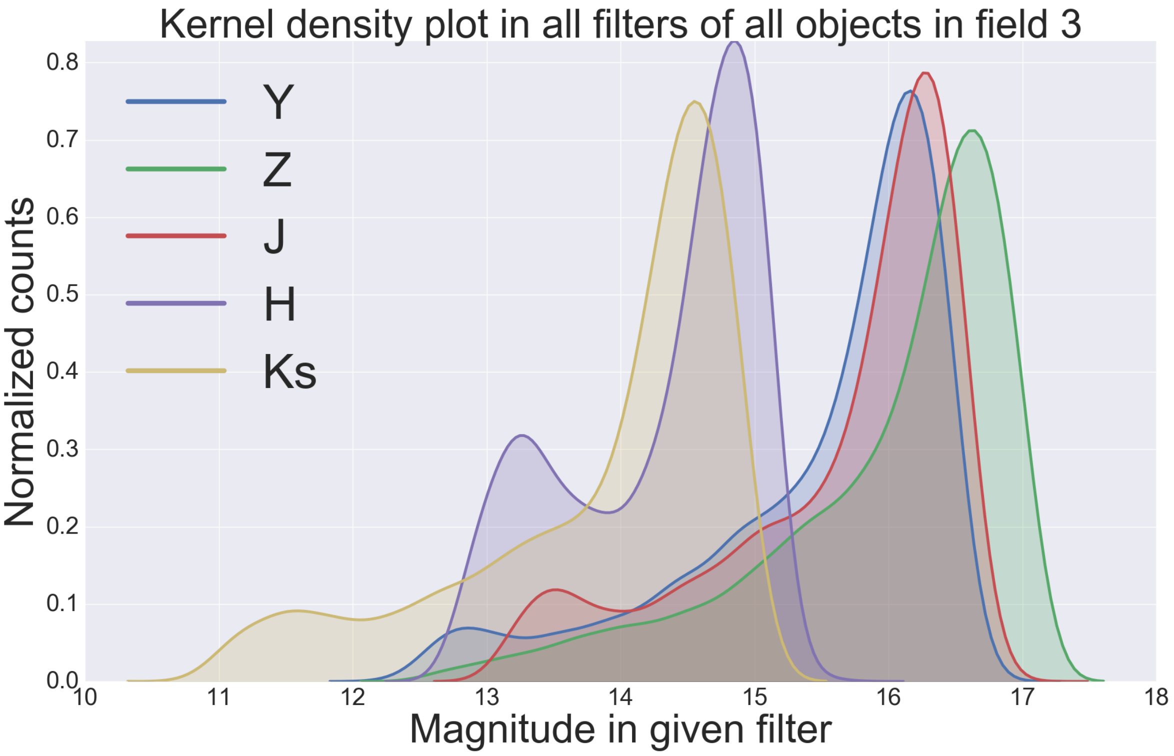

Figure 5: Kernel density plot of all magnitudes in field 3.

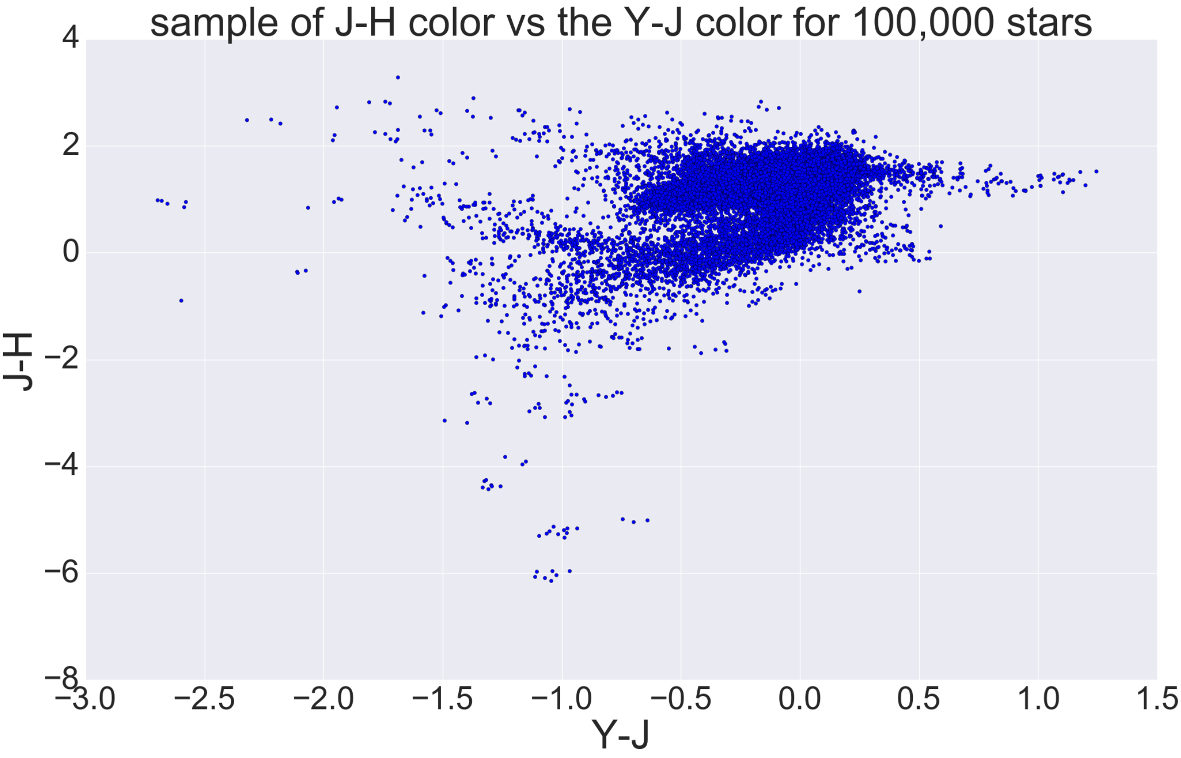

Figure 6: Scatter plot of the sample of 100,000 stars created from the kernel density distribution of the objects from the database.

b) Linear regression

The next steps in the process will be the training and the application of the regressor. There are various ways to do this and I will start with linear regression. There are three main types of linear regression estimators: ridge, LASSO and linear regression. When the amount of datapoints N is comparable to the amount of parameters p, linear regression typically has a large variance. Ridge and LASSO regression combat this by trading a bit of the bias for a lower variance. In our case, however, N is much larger than p and we expect Ridge and LASSO regression to converge to linear regression. Let's still use all three types of linear regression to see whether the results agree with this expectation. Using the whole of file A for training we obtain the training errors that are displayed in Table 3. In the case of ridge and LASSO regression you have to specify a value for α in the *sklearn* modules. This α is the parameter that balances the amount of emphasis given to minimizing RSS versus the minimizing sum of square of coefficients.[1] To determine the best value of α, I looped over a lot of values and checked which one works best. The smaller the value of this α, the closer ridge or LASSO regression converges to linear regression. Since the values of α are pretty small (the value for LASSO is even very small), our expectation that ridge and LASSO regression converge to linear regression seems to be valid. Therefore it does not make much of a difference which regression method we use, so if I would have to choose I would go for the simplest one: linear regression.

Table 3: Comparison of the training errors of the different types of linear regression

Regression type	Training error	Value of α
Linear regression	0.01455886	-
Ridge regression	0.01455690	0.1905
LASSO regression	0.01455853	8×10^{-7}

c) Training error as generalisation error

The error we estimated on the training sample is not a reliable estimator of the generalisation error of. The reason for this is that the model was calibrated on the same data the error is based on. To get an honest quantification of the generalisation error we need to estimate the error obtained from another dataset, one that has nothing to do with the calibration of the model. For this purpose we can use file B. The results of the honest error estimation of the linear regression models are shown in Table 4.

Table 4: Comparison of the test error obtained by different types of linear regression

Regression type	Test error	Value of α
Linear regression	0.01460145	-
Ridge regression	0.01460299	0.1905
LASSO regression	0.01460557	8×10^{-7}

d) Other regression methods

Next to the linear regression models discussed above there are also quite some non-linear methods available for the training and application of the classifier. Since it is hard to say which method will perform best on a particular dataset, let's explore the *sklearn* package and try some of the available non-linear methods. The results are shown in Table 5, which also shows the used parameters. Some regression methods perform namely different when a certain parameter is changed. I have determined the best parameter value by using 10-fold cross-validation on the training set (file A). With the best parameter value I run the model on the test set (file B) to get the generalized error.

[1] www.analyticsvidhya.com/blog/2016/01/complete-tutorial-ridge-lasso-regression-python

Table 5: Comparison of the test error obtained by different types of non-linear regression

method	Estimated error	used parameter
k-nearest neighbors	0.01246486	k = 19
Random forest	0.01193203	97 trees
Gradient boosting	0.0116081	315 boosting stages
Multilayer Perceptron (neural network)	0.01426410	hidden layer size: 130

e) Advantages/disadvantages relative to linear regression

Since the gradient boosting regressor turns out to give the lowest generalization error (although it is just a slight favourite over k-nearest neighbours and random forest regression) I would choose this type of regression as the best method. The most obvious advantage of this method compared to linear regression is that the generalized error is lower, giving a better prediction of the redshift from photometry. The most obvious disadvantage is that it is computationally expensive, especially when using boosting stages up to 315. Another disadvantage is the danger of overfitting. Since linear regression is a very simple model, overfitting will not be a problem. For non-linear regression one should always watch out for overfitting, although according to the *sklearn* documentation a high amount of boosting stages is encouraged since gradient boosting is fairly robust to overfitting.

Kernel distribution in J-H,Y-J space of the sample of 100,000 stars

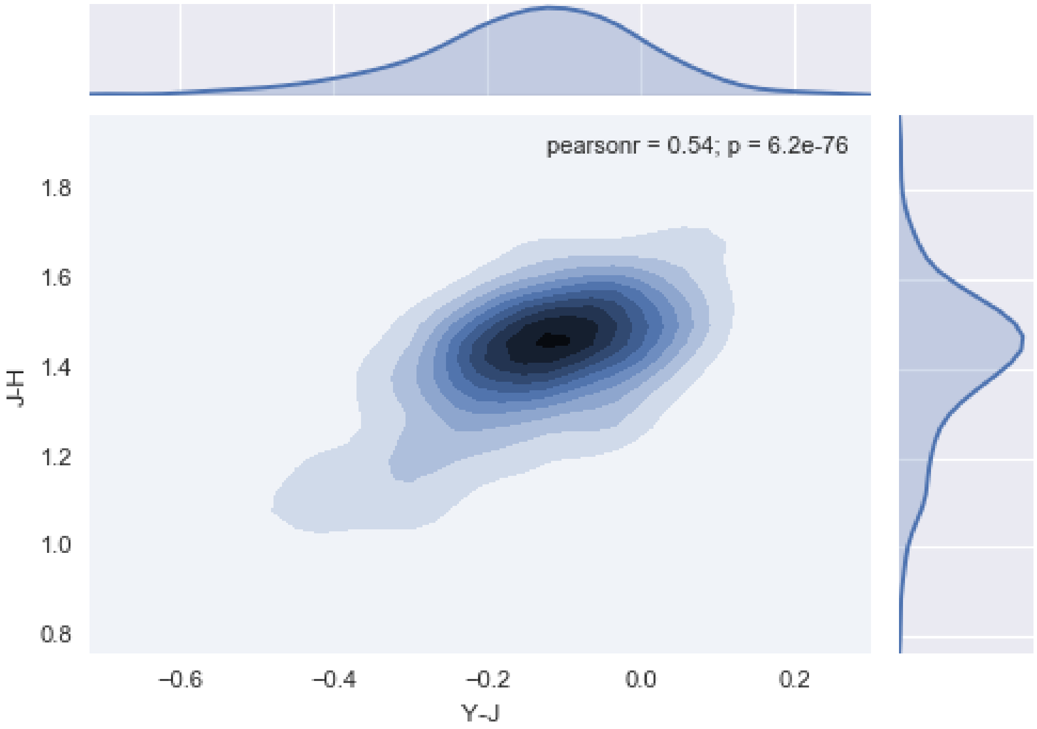

Figure 7: Kernel density plot of the sample of 100,000 stars created from the kernel density distribution of the objects from the database. The pearsonr indicates that there is some correlation between Y-J and J-H, but this is not important for this purpose. The p-value indicates the probability of the system producing a dataset that have a pearsonr of at least the given value.

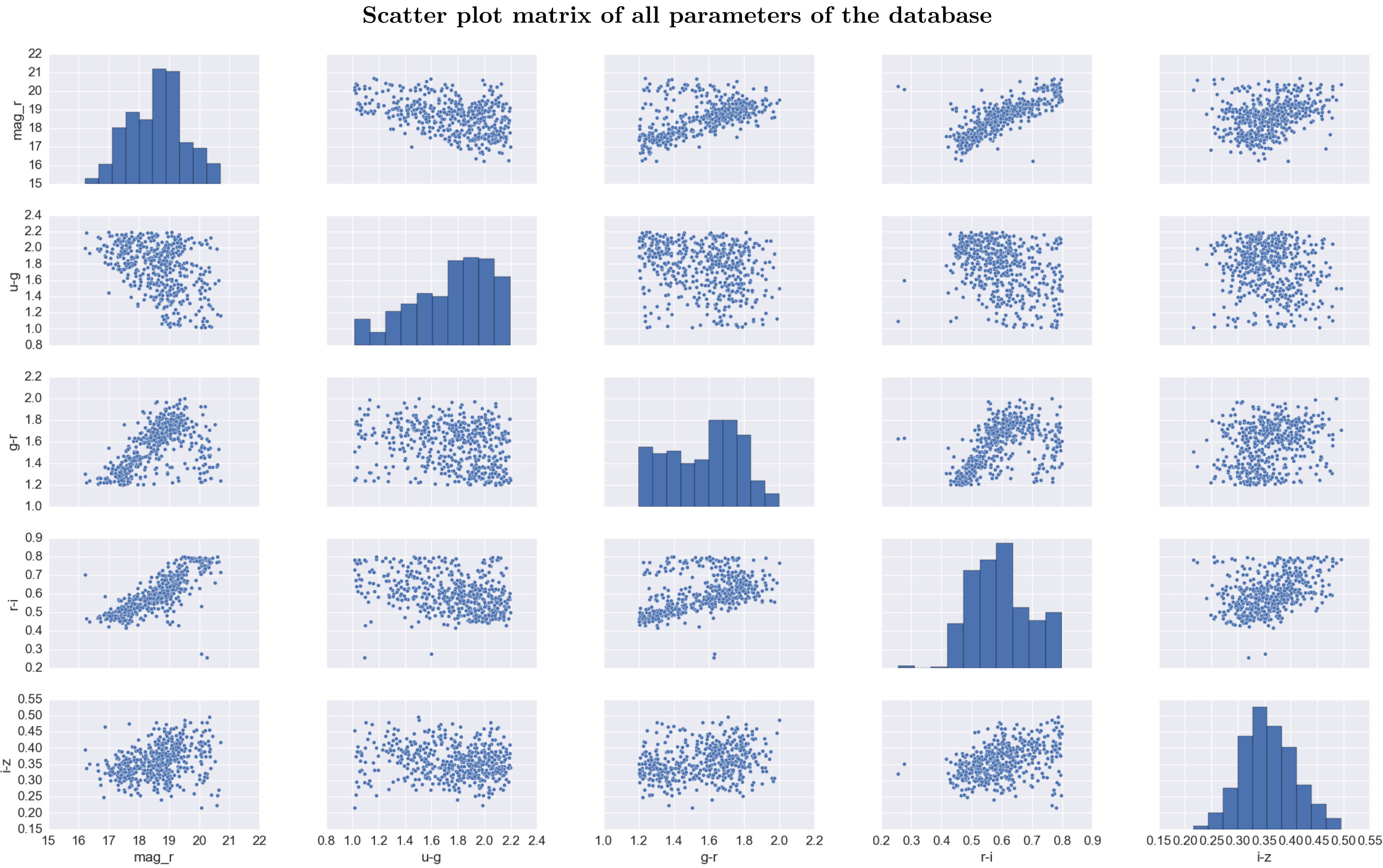

Figure 8: Scatter plot matrix of all parameters from the database. For clarity only 500 datapoints are plotted. Some linear relation can be recognized between the r magnitude, the g-r colour and the r-i colour.