Data Bases and Data Mining in Astronomy

Tom Sweegers (s1519530)

University Leiden, 26 January 2018

Abstract: In this report, I present the methods used to complete the exercises of the final problem set for the course "databases and datamining in astronomy" (DBDM). There are two main problems. The first problem is about the creation of a database and working with it. In the second problem, machine learning techniques are used to find a relationship between the color and redshift of galaxies. With this report come several python codes and the database created in section 2. These files can be found in the github repository (see section "appendix").

1. Introduction

Astronomy has changed in the past decades. It's exponential growth of data values, data complexity and data quality[3] changes the once data-poor science into a data-rich. Time-domain astronomy plays a major role in this process. It studies the change in time of astronomical objects. In order to do that, a large number of data files are generated to keep track of the objects. The growth of time domain astronomy and its data come with new facilities such as DPOSS[2], 2MASS[8] and SDSS[9]. Also the MASCARA[6] project in Leiden plays a role in this process. With all these amounts of data, new problems arise. The rate and volume at which new data is provided is too high to have humans analyze it. The process of analyzing data has therefore been taken over by computers. Robotic telescopes automate the analyzing part by reducing the data and placing it in ordered databases before uploading it to workplaces like the cloud[3]. In section 2, a way to create databases to hold these data is described.

In section 4, the estimation of the redshift from galaxies that are observed with photometry is discussed. The analysis of data from photometric surveys is more useful when the properties of galaxies are known. Spectroscopic analysis is the most accurate measurement to calculate the redshift of galaxies. But this requires large telescopes and is also time consuming [10]. The redshift is therefore attempted to be estimated by the use of the measured photometric properties. This is done by using a part of the data as training sample. With this training sample, an algorithm is constructed to map the galaxies colors onto its spectroscopic redshift. The algorithm is then used as a function such that the photometric redshift is estimated from the colors of a galaxy. This process is called machine learning. Several techniques are considered in this report while aiming for the lowest training error. Connolly et al.[1] used linear regression to find the photometric redshift of 370 galaxies up to a redshift of $z = 0.5$. While Jouvel et al.[5] made use of non-linear random forest regression and Zitlau et al.[10] used boosting techniques.

2. Database for time-domain surveys

The setup of the database is made by the program DB_Creation.py. The code is based on Brinchmann's make_tables_python.py where a simple database with two tables is made. In the DB_Creation.py file, a table for file_info_for_problem.csv is made first. In this table, the Julian Date, air mass and exposure time are given for the fits files. The fits files contain the measured data of the time-domain survey in different filters and fields. The code grabs all fits files in the current directory with:

```
fits_files = glob.glob('*.fits')
```

A loop over this list of fits_files then produces a table in the database with information on right ascension and declination of the objects in decimal degrees, the x and y location of the objects in pixels, the fluxes with different aperture diameter and their uncertainties as well as their calibrated magnitudes and the uncertainties on this. Nan values in the tables are replaced by NULL in the database. Once the database (called Final_Project.db) is created, it can be opened in sqlite3 by the following command:

```
sqlite3 Final_Project.db
```

In the next subsections, some scientific questions are answered using SQL commands on the database.

2.1 Find images observed between two MJD's

The Julian dates of the observations are provided in the file_info_for_problem.csv table. A column containing the filename of the corresponding fits file is added to the table. The SQL command to find the filenames observed between Julian dates 56800 and 57300 reads:

```
SELECT Filename
FROM file_info
WHERE MJD
BETWEEN 56800 AND 57300;
```

2.2 How many stars are detected with S/N > 5

Since the flux and the uncertainty on this flux is given in the tables. We can calculate the signal to noise ratio (S/N) by dividing the flux (signal) with the uncertainty on the flux (noise). In all of the following SQL commands, I will be using the Flux1 and the Mag1 columns with aperture diameters of 1", since the bigger aperture diameters might contain too much background. Counting the number of objects in a fits file (for example Field_1_H) that matches the criterion of having $Flux1/dFlux1 > 5$ is then done using:

```
SELECT COUNT(*)
FROM Field_1_H
WHERE Flux1/dFlux1 > 5;
```

But the data does not only contain stars. A column giving the type of object is present in the fits tables. Stars are given a class number equal to -1. This is used to find out if an object is a star or not by adding the line:

```
AND Class = -1;
```

To combine this criteria with the criteria of the Julian date from section 2.1, we need to be able to compare the tables on the filename. In SQL it is not possible to compare the name in a column, to the name of other tables. This can be done in PYTHON language rather easy, but since we want to do all these criteria fully in SQL, we will have to add a column to all tables stating their table name. The problem when doing this is that unnecessary data is added, meaning that the database increases in size. In this case, adding an extra column to all fit tables means an increase of almost 10 percent in size. I consider this not worth doing when there are other options available. For instance, executing the command to count the number of stars for each file given by the MJD command separately. But the most efficient solution is just using python. The computation costs less space and it is computed faster than executing the same command for each file, especially when there are a lot of files between those Julian dates. The search query in python is done in a new program called DB_Analysis.py. The result of the combined search queries is given below:

$Field\_1\_Z$	9862	$Field\_2\_H$	9985
$Field\_1\_J$	9985	$Field\_2\_Y$	9985
$Field\_1\_H$	9990	$Field\_3\_Z$	9930
$Field\_1\_Ks\_E003$	9992	$Field\_3\_J$	9983
$Field\_1\_Y$	9967	$Field\_3\_H$	9987
$Field\_2\_Z$	9968	$Field\_3\_Y$	9977
$Field\_2\_J$	9985		

2.3 Find objects that have J-H > 1.5

Each fit file contains data observed in a single filter. The objects in the files can be compared with each other by their StarID (a column given in the fit tables). The tables are thus joined on the StarID. The flux from the J tables minus the flux from the H tables is calculated and checked if it exceeds 1.5. The result for the first 20 objects in the Field_1 tables is produced after the following command:

```
SELECT H.StarID, J.Mag1-H.Mag1
FROM Field_1_H as H
JOIN Field_1_J as J
ON H.StarID=J.StarID
WHERE J.Mag1-H.Mag1>1.5
LIMIT 20;
```

But there are two other fields with J and H filter, and there are a lot more than 20 objects with J-H color bigger than 1.5. To visualize the data, a python script is added to DB.Analysis.py containing this SQL code and the plotting of the StarID versus the J-H color. This plot is showed in figure 1.

2.4 Find objects that differs more than 20 times the flux uncertainty from the mean flux

For this question, the tables from the observations done in the Ks filter are joined together. This is done separately for each field. After joining, the mean flux for each StarID and the mean uncertainty on the flux is calculated. Since both have the same number of observations, the 'divided by the number of observations' can be left out so that the fluxes and their uncertainties are just added up. We multiply the uncertainties by 20 and look for StarID's where this is bigger than the sum of the fluxes.

```
SELECT Ks1.StarID
FROM Field_1_Ks_E001 as Ks1
JOIN Field_1_Ks_E002 as Ks2
ON Ks1.StarID=Ks2.StarID
JOIN Field_1_Ks_E003 as Ks3
ON Ks1.StarID=Ks3.StarID
WHERE (Ks1.Flux1 + Ks2.Flux1 +
Ks3.Flux1) < (Ks1.dFlux1 +
Ks2.dFlux1 + Ks3.dFlux1) * 20
LIMIT 20;
```

There are again a large number of objects that hold the relation of having mean flux bigger than 20 times the mean uncertainty. Therefore, the results are again provided graphically using python (see figure 2).

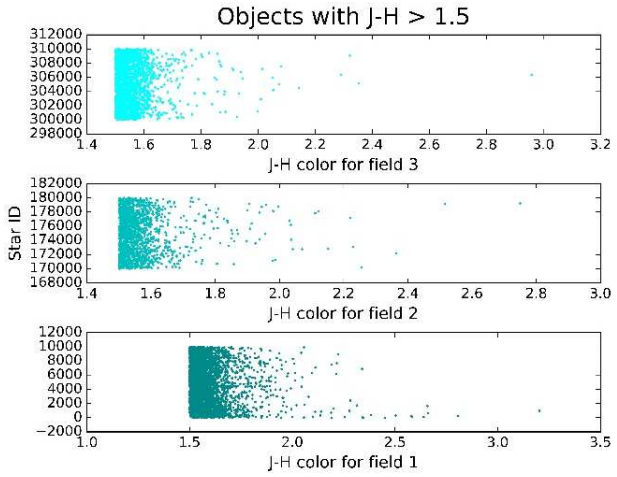

Figure 1: Visual representation of all objects that have a J-H magnitude bigger than 1.5. The three fields contain objects with very different Star ID's. Each field therefore has its own figure.

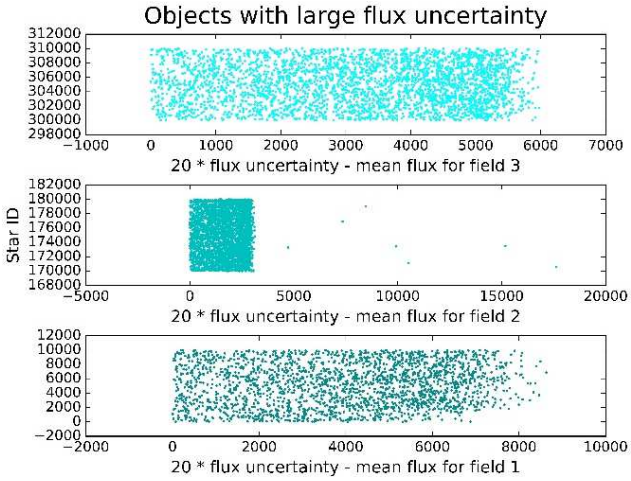

Figure 2: Visual representation of all objects where Ks differs by more than 20 times the flux uncertainty from the mean flux.

2.5 Find all tables that exist for a given field

When creating the database, the filenames from the original file_info_for_problem.csv were changed into the true filenames of the fits tables. The names of the catalogues with a certain FieldID are therefore easily obtained. The command that searches for the fits tables with FieldID 1 is:

```
SELECT Filename
FROM file_info
WHERE FieldID=1;
```

Where the result provided, is:

$Field\_1\_Z$	$Field\_1\_H$	$Field\_1\_Ks\_E002$
$Field\_1\_J$	$Field\_1\_Y$	$Field\_1\_Ks\_E001$
		$Field\_1\_Ks\_E003$

2.6 Retrieve Y, Z, J, H and Ks magnitudes for all stars that have a S/N > 30

To get all different magnitudes in a certain field, all tables need to be joined on their StarID column (like section 2.3). In section 2.2, the signal to noise ratio and the class number of stars are already used. The Class of an object with a certain StarID is always the same no matter what filter you look at. But the S/N ratio is not. To be certain that the star has a signal to noise ratio bigger than 30 in every filter, one could use the WHERE statement on every filter, but for simplicity I only use the Y filter assuming the S/N does not vary to much in different filters. Suppose we look at field 2, the SQL command looks as follows:

```
SELECT Y.StarID, Y.Mag1, Z.Mag1, J.Mag1,
H.Mag1, Ks.Mag1 FROM Field_2_Y as Y
JOIN Field_2_Z as Z ON Y.StarID=Z.StarID
JOIN Field_2_J as J ON Y.StarID=J.StarID
JOIN Field_2_H as H ON Y.StarID=H.StarID
JOIN Field_2_Ks_E001 as Ks
ON Y.StarID=Ks.StarID
WHERE Y.Flux1/Y.dFlux1 > 30
AND Y.Class = -1
```

Figure 3 shows the results of this query. The scatter plot is made in the python program DB.Analasis.py.

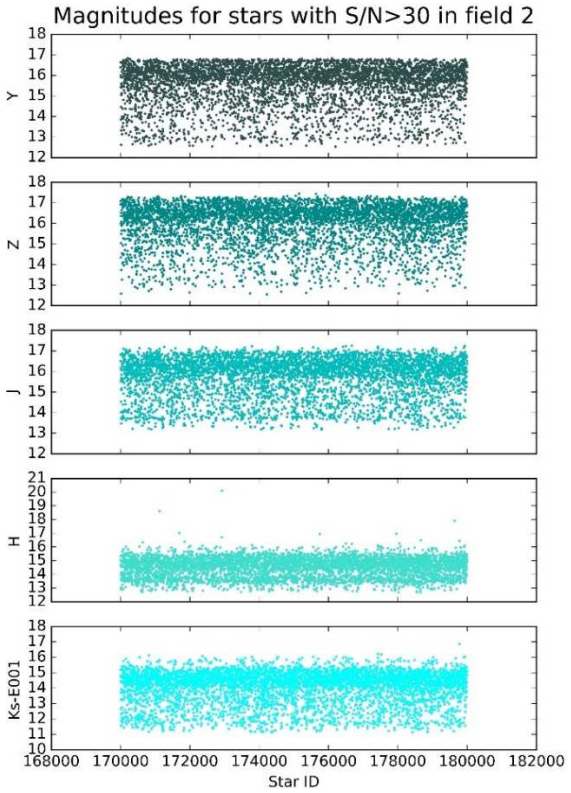

Figure 3: The Y, Z, J, H and Ks Magnitudes from field 2 for all stars that have a S/N ratio larger than 30.

3. Simulation Sample

The database can be used to create a sample of simulated stars for the Euclid space mission. Therefore, the distribution of the stars in Y-J and J-H magnitudes will be used. A new sample of 100,000 stars is aimed for.

To do this, a python program DB_Sample.py is created. To get the Y-J and J-H magnitudes from the different tables in python arrays, the following SQL code is used:

```
SELECT Y.Mag1-J.Mag1, J.Mag1-H.Mag1
FROM Field_"+str(i+1)+"_Y as Y
JOIN Field_"+str(i+1)+"_J as J
ON Y.StarID=J.StarID
JOIN Field_"+str(i+1)+"_H as H
ON J.StarID=H.StarID
WHERE Y.Class=-1;
```

This SQL code is placed inside a 'for'-loop where the parameter i goes from 0 to 2 (for three different fields). The query results for the three fields are added to a single list. The distribution for all these stars in Y-J and J-H magnitudes is showed in figure 4.

The 100.000 stars could be fitted with a multivariate Gaussian (normal) distribution. But this means, that the stars in the database should also follow this distribution. One way to classify what distribution function the data follows, is by graphical analysis of the histogram plot of the data. The histogram is 'Bell'-shaped for a Gaussian. Another possibility would be a Laplace distribution (sharply peaked) or an exponential distribution. Since the data is a two dimensional distribution, there are two histograms (see figure 7). The upper plot shows the distribution of the x-axis (Y-J magnitudes) from figure 4. The lower plot shows the distribution of the y-axis (J-H magnitudes) from figure 4.

Both histograms of Figure 7 show the desired bell shape forms that resemble a Gaussian distribution. Notice that at the left side of the medians, there seems to be another Gaussian function hidden. The unknown distribution thus seems to be the sum of multiple Gaussian distributions. The means, medians and standard deviations for these Gaussian functions can be calculated using Gaussian Mixture Modelling. I used the python package GaussianMixture from sklearn.mixture to do this.

The sample of 100.000 stars can now be simulated using the same Gaussian distribution function as the data. Figure 5 shows the distribution of the sample simulated with Gaussian Mixture modelling using 20 Gaussian distributions. Figure 8 shows the corresponding histograms for the Y-J and J-H magnitudes of the x and y axes of figure 5. These figures look very similar to those of the data (figures 4 and 7). The distribution where the sample comes from, is thus approximately the same as the distribution for the stars from the database. But trying to fit the histogram with 20 Gaussian distribution most likely gives a problem: Overfitting. Overfitting means that the model is to flexible and is not a general solution (for a different dataset, the model will not fit the data at all).

To be sure that overfitting is not a problem, I reduced the number of Gaussians to 5. The distribution of the new sample fitted with 5 Gaussians is showed in figure 5. The histogram plots for the Y-J and J-H magnitudes of this sample is showed in figure 9. These figures are still similar to the figures of the data and this 100.000 star sample can therefore be said to be of the same distribution as the stars from the database.

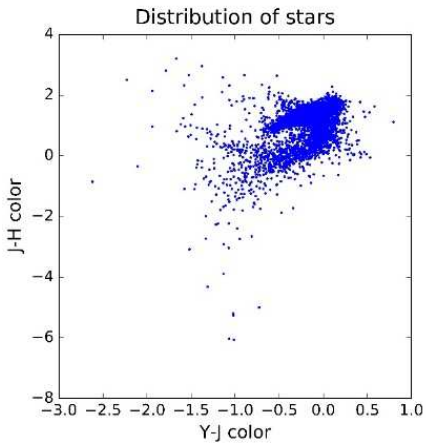

Figure 4: Distribution for all the stars present in the database.

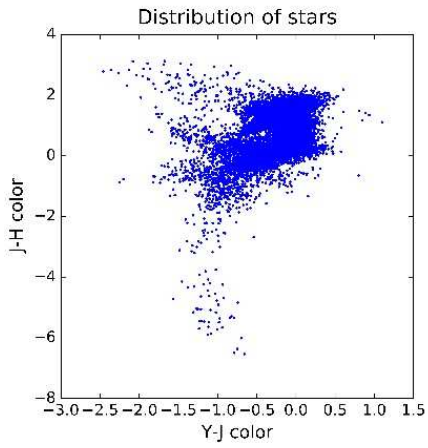

Figure 5: Distribution for the simulated stars overfitted with 20 Gaussians.

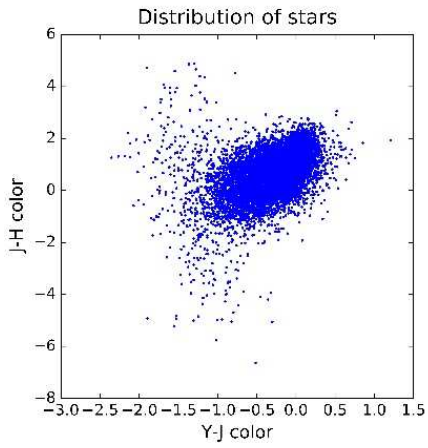

Figure 6: Distribution for the stars simulated with 5 Gaussians.

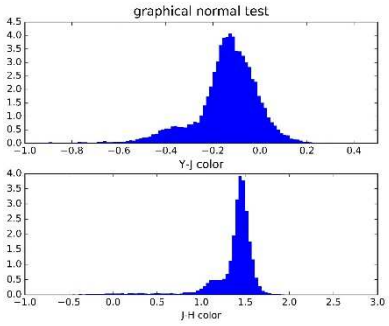

Figure 7: Histogram plots for the Y-J magnitudes (upper plot) and J-H magnitudes (lower plot) of the stars from the database.

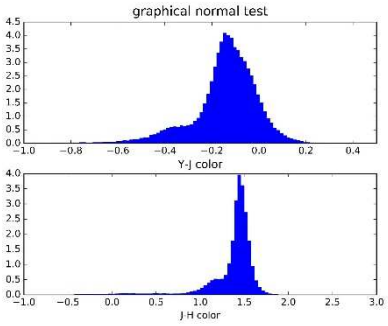

Figure 8: Histogram plots for the Y-J magnitudes (upper plot) and J-H magnitudes (lower plot) of the simulated stars that are overfitted with 20 Gaussians.

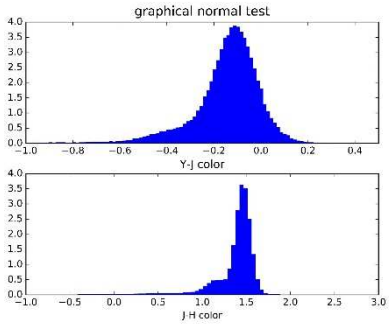

Figure 9: Histogram plots for the Y-J magnitudes (upper plot) and J-H magnitudes (lower plot) of the 100.000 simulated stars fitted with 5 Gaussian distributions.

4. Photometric redshifts of galaxies

Astronomical observatories can be divided into two categories: spectroscopic and photometric redshift surveys. Spectroscopic redshift observatories are used for galaxy clustering while photometric redshift surveys detect gravitational lensing, clusters and clustering[5]. Spectroscopic data can be used to accurately determine the redshift of a galaxy. Photometry data is needed to estimate certain properties of galaxies such as its colors or stellar masses. But since the determination of spectroscopic redshifts requires large telescopes and is time consuming [10], photometric redshifts are estimated using the measured photometric properties. In this section, two files (PhotoZFileA.vot and PhotoZFileB.vot) are provided (see the appendix). These files contain information on the spectroscopic redshift and four different photometric colors of over 70.000 galaxies each.

With large data sets, classification methods or fitting tasks can overfit training samples. Which means that the model is to flexible and is not a general solution (with different training sets, the model will likely not fit the data). A way to improve this is by using feature extraction. This reduces the number of required variables that describes a large data set. It is used for dimension reduction. Another advantage is that the storage space can be reduced. A technique to do this is by principle component analysis (PCA). It maps the data linearly onto a lower dimensional space such that the variance is maximized. A consideration that should be made before using it as a classification or fitting task, is that dimension reduction methods are sensitive to outliers and deal badly with noisy and/or gappy data.

4.1 Linear Regression Techniques

Linear regression makes a model by fitting a linear function to the data. Ridge regression uses cross validation to learn the parameter as well. It will then minimize the impact of irrelevant features from the data. Lasso regression has the regularization term in absolute values. By doing this, it will not only give irrelevant data points a lower impact, but it gives them no impact at all by setting them to zero. An advantage of this, is that there are fewer features included in the model.

The question one should ask then, before using a regression technique, is if you want to penalize some of the data points so they have less influence on the model, and if you do, will you ignore those data points?

I'll be using Linear regression without scaling the impact of badly fitting points. The reason for this is that the training data PhotoZFileA.vot has information over more than 70.000 galaxies. The effect of badly fitting points is therefore minimal. After fitting a linear function to the training model (file A), PhotoZFileB.vot is left to evaluate if the model works on other datasets as well. Comparing the estimation error from file A with that of file B could be a warning for overfitting problems. If this is the case, then file A would have a much lower error than file B.

Before fitting a model, a multidimensional plot of the u-g, g-r, r-i and i-z colors and the spectroscopic redshift is made. This figure can be used to explore the relations between the colors and the redshift. The creation of this plot, but also the fitting of the linear model and the non-linear model in section 4.2 is done by the python code Photometric_Redshift.py. The plot is showed in figure 10.

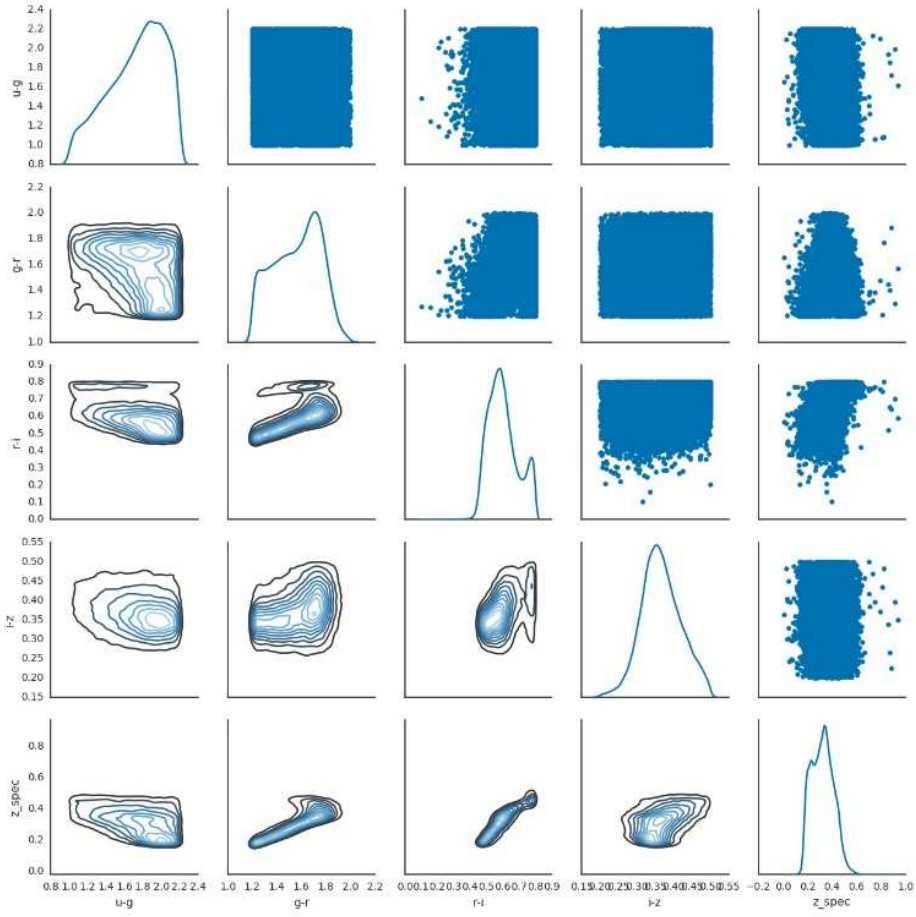

Figure 10: Multi-dimensional data graph of the four color magnitudes and the spectroscopic redshift from PhotoZFileA.vot. The diagonal figures are kernel density estimations (KDE's). These are smoother versions of a histogram. Figures below the diagonal are made by 2D histograms. Figures above the diagonal show the relations between variables using scatter plots.

It is clear from figure 10, that there is no specific relation visible between a certain color and the spectroscopic redshift. Linear regression is therefore applied on all four colors. The relation that the LinearRegression package from AstroML returns is:

$$\begin{aligned} Z_{phot} = & -0.218 - 0.027\,(u-g) \\ & + 0.115\,(g-r) \\ & + 0.674\,(r-i) \\ & + 0.007\,(i-z) \end{aligned} \qquad [1]$$

This is seen as the best fitting linear model. To find out how well it predicts the photometric redshift, the predicted redshift is compared with the given spectroscopic redshift. A value close to 0.014 is asked for since this is the lowest training error possible. The error is computed by the normalized median absolute deviation (NMAD). It is given by[4]:

$$E(z_{phot}) = median\left(\left|\frac{z_{spec}-z_{phot}}{1+z_{spec}}\right|\right) \qquad [2]$$

A training error of $E(z_{phot}) = 0.0151$ is found. This is reasonably close to the minimal value. In section 4.2, I consider non-linear regression methods to predict the photometric redshift. With that, a lower generalization error than the linear regression method is aimed for.

In general, the training error is not reliable when computed on its own training sample. The photometric redshift is of course close to the spectroscopic redshift because the model function was made to be as close as possible to that value. When the model is tested on a different training sample, one can find out if the model is a good representation of the relationship between the colors and the redshift by computing that estimated error. To do this, PhotoZFileB.vot is used as the training sample. The photometric redshift is estimated with the linear relation given in equation [1]. The given spectroscopic redshift is then used to compute the training error of equation [2]:

$$E(z_{phot}) = 0.0152 \qquad [3]$$

Equation [3] shows the result of the training error when using file B. Since this is not far away from the training error computed on file A itself, the model given by equation [1] can be said to be a good approximation of the photometric redshift.

4.2 Non Linear Regression Techniques

Non-linear regression minimizes the residual sum of squares (RSS) for the distances of the data to the modelled curve. This is just like linear regression. But non-linear regression cannot solve this in one step. The model is made iteratively. An initial estimate of the value of each parameter must be provided. The model then adjusts these values to improve the fit of the curve to the data[7].

K-nearest neighbor (KNN) regression is a non-linear regression technique that searches for local trends in the data. It calculates the mean of the K nearest points and gives every point the same weight. As with linear regression and ridge regression, if the weight of the points is variable, KNN regression is called Kernel regression instead. KNN regression is used mostly as a reference or for quick exploration of small datasets. Kernel regression is a better alternative when the dataset is moderately sized. Since our dataset has over 70.000 galaxies, both methods are not optimal and an alternative method needs to be used.

Random forest regression is a combination of decision trees with bagging. A decision tree is a model that predicts the value of a variable by the input parameters. The data is stepwise divided according to some criteria. Bagging helps to avoid overfitting problems. Bagging starts by drawing a bootstrap sample from the data. Meaning that bagging makes n new training sets. These n training sets are used to generate n models for the data. The final step of bagging is to combine these results to one model by averaging the results.

Boosting methods combine many weak learners (simple algorithms that are only slightly correlated to the true model), in order to obtain a strong learner (represents the true model well). Every step of the process, boosting fits a weak learner to the data. Data points that do not fit that current model are given a higher weight in the next step.

Another option can be, by using neural networks. These computer algorithms are designed to work like a human brain would. The algorithm learns from examples. Input data given to the neural network is processed by (several) activation functions after which the output data is given. The advantages of neural networks is that it is simple to use and that they can learn any function with the help of example data. But it is hard to understand what is happening in the activation functions.

I will be using the neural networking technique. This is provided by the MLPRegressor package from sklearn.neural_network. The LBFGS solver of this regressor works well for small datasets since it can converge faster. But the default ADAM solver is better for relatively large datasets (more than 1000 training samples). Therefore, the ADAM solver is more suitable.

I computed the photometric estimates several times and found out that the estimated errors for file A and B are varying. On average, after estimating the photometric redshift 10 different times, I found:

$$E(z_{photA}) = 0.0136 \quad [4]$$

$$E(z_{photB}) = 0.0137 \quad [5]$$

Where the lowest errors in these 10 simulations were 0.0126 and 0.0127 respectively. None of the simulations got a higher error than the estimated errors in the linear case. The difference between the estimated photometric redshift and the given spectroscopic redshift is visualized in figure 11. In the optimal case of having an error of $E(z_{phot}) = 0$, the points would all lie on the red line.

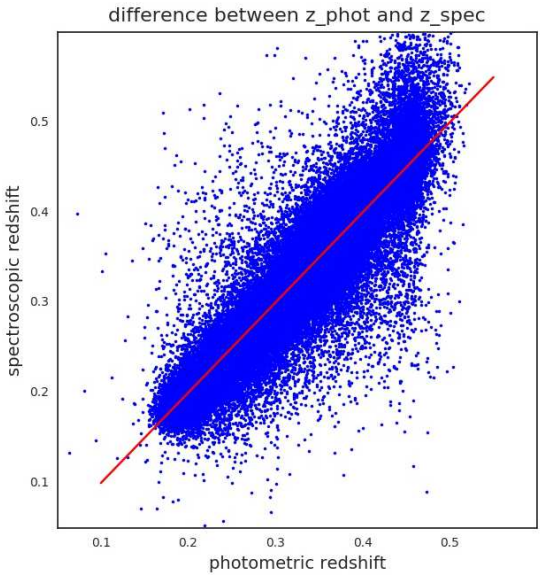

Figure 11: Blue dots indicate all objects from PhotoZFileA.vot. Their spectroscopic redshift is given in the file. The photometric redshift is estimated using neural networks. The red line indicates the optimal solution. On this line, all photometric redshifts are predicted to have exactly the redshift obtained from spectroscopy.

5. Conclusions

For the final problem set of the course databases and datamining in astronomy, I've made a database for time domain astronomy. The setup of the database is made, such that search queries for the database can be done mainly in SQL code.

The distribution of stars from the database is used to create a new sample of 100.000 stars for the Euclid space mission. The new sample is made using multivariate Gaussian mixture modelling. The mixture modelling searched for a sum of 5 Gaussian distributions to prevent overfitting the data.

The second part of the assignment was to evaluate linear and non-linear regression techniques. These techniques were used to make predictions for the redshift using only photometric data. Spectroscopic redshifts were given for the objects such that a training sample could be used. Linear regression turns out to be nice in use because of its simplicity. But linear regression fails when one wants to expand to higher redshifts, or fainter magnitudes [(1)]. Linear regression is also less accurate in this case then Non-linear regression. This is due to the relation of the colors not to be exactly linear with the redshift. The neural network regressor therefore got a lower training error. Neural network regression on the other hand gives very little insight in the activation functions. This could cause the production of misleading results [(7)]. The function that was fitted, produced different results every time it was ran. The most optimal solution is therefore not always found. Linear regression always finds the same function as the best fit (although this function has a higher estimated error than the neural networks).

References

[(1)] Connoly A.J., et al.: 1995a, Astron. J. 110, 2655

[(2)] Djorgovski S.G., et al. (DPOSS): 1998, Wide Field Surveys in Cosmology, Editions Frontieres, 89

[(3)] Graham M.J., et al.: 2012, DaPD, 30, 371

[(4)] Ilbert O., et al.: 2009, ApJ, 690, 1236

[(5)] Jouvel S., et al.: 2017, MNRAS, 469, 2771

[(6)] Lesage A.L. et al.: 2014, SPIE, 9145, 8

[(7)] Motulsky H.J., Ransnas, L.A.: 1987, FASEB J. 1, 365

[(8)] Skrutskie M., et al. (the 2MASS team): 2006, AJ, 131, 1163

[(9)] York D.G, et al. (the SDSS team): 2000, AJ, 120, 1579

[(10)] Zitlau R., et al.: 2016, MNRAS, 690, 3152

APPENDIX

Besides the articles from the references, this article has used documents provided by Jarle Brinchmann in the 2017th course Databases and Datamining in astronomy. All course materials can be found in the github repository:

https://github.com/jbrinchmann/DDM2017

Data and python programs as well as the images in this article and the database created in section 2 can be found in the following github repository:

https://github.com/TomSweegers/DBDM_Final_Project