Data bases and data mining project report

Prof. Jarle Brinchmann
Juan Manuel Espejo **ID.** s1852477
Leiden Observatory

January 26, 2018

Introduction: The course of data bases and data mining in Astronomy taught us a lot of computational tools to understand and analyze data in a scientific way. This specific project involves the implementation of the two biggest components of the course which are the creation and usage of databases with Python and SQL and the manipulation through statistics, regression, etc. of data in the context of our usual astronomical research. This report is divided into these two main components and in each of them I make reference to the jupyter notebooks with the codes for every section. For facility in the reading, the SQL queries are specified in the appendix and the discussion is done on the text for every section as well.

Problem 1: A DATABASE FOR A TIME-DOMAIN SURVEY

—>Follow the jupyter notebook<—
`Database_and_queries.ipynb`

1.a Creation of a schema for a Database

This part of the project has to do with the design of a database and the manipulation of its data in a rather realistic way for Astronomy. The database is based on a survey with observations of a patch of the sky at certain intervals. The filters used in this survey are Z,Y,J,H and multiple observations in the Ks band to construct light curves of stars and check their variability.

Our initial package of data consists of multiple files of different fields and observations in multiple filters and we must construct a database that contains all the information that allows the user to properly use the data for scientific purposes.

By looking at the data we see that there are three fields of observation with multiple tables each, this means that all objects in the survey are probably observed more than once and all of them will have a characteristic ID. Noting this, we can put all the data in one table where it would be possible to retrieve all the information about one specific object at any time (by using it's respective ID).

For further analysis and handiness in handling the data, and noting that not all the objects have all measured magnitudes in all the filters I also decided to create a table with certain colours. So I decided to lay out my database with the creation of three tables in total, the first one is a table with the information of the files and their matched names which I name "info", the second one is a big data table with all the information of all the objects and it's named "All_data" and the third one is a table that contains the colours JH and YJ of the starts that were observed in those filters. That table also contains the ID and field ID of the stars for further processing and its name is "Colours".

The first thing I do is solve the ambiguity of the names of the .fits files with the names given in the information file because they don't match. This is easily done by reading off the Julian date on the long names of the files and match them with their corresponding short name and then adding another column to the "info" file with the identified name; I call this column "name_file" and it allows me to properly identifiy the files with their filter and their field e.g Field-1-Z corresponds to the file Z-ADP.2017-01-18T11:58:36.905.fits. The schema (as done in the lectures and based on [1]) of this table is shown in Table 1.

Column	**Type**	**SQL Type**	**Other**
ID	Integer	INT	1-18
FieldID	Integer	INT	1-3
Filename	String	VARCHAR(50)	Long name
Filter	String	VARCHAR(5)	Z,J,H,Y,Ks
MJD	Float	DOUBLE	
Airmass	Float	DOUBLE	
Exptime	Float	DOUBLE	
name_file	String	VARCHAR(50)	Short name

Table 1: Schema lay out for the table "info". It specifies the structure and the the type of information in all the columns of the table.

Now that my "info" file is ready, I can build my main table by stacking all the information from the .fits files and using the ID of the fits files, the filters and the field ID as identifiers. I also add an additional column to my table which is the signal to noise ratio calculated as the ratio between the value of Flux1 and dFlux1. I call this column "SN". The schema of this big table is shown in Table 2.

Column	**Type**	**SQL Type**	**Other**
RunningID	Integer	INT	For every object
X	Float	DOUBLE	In pixels
Y	Float	DOUBLE	In pixels
Flux1	Float	DOUBLE	ADU
⋮	⋮	⋮	⋮
dFlux3	Float	DOUBLE	ADU
Ra	Float	DOUBLE	Radians
Dec	Float	DOUBLE	Radians
Class	Float	INT	Flag
Mag1	Float	DOUBLE	ADU
⋮	⋮	⋮	⋮
dMag3	Float	DOUBLE	ADU
StarID	Integer	INT	
ID_fitsfile	Float	INT	
Filter	String	VARCHAR(5)	Z,J,H,Y,Ks
FieldID	Integer	DOUBLE	1-3
SN	Float	DOUBLE	Flux1/dFlux1

Table 2: Schema for the table "All_data". The vertical dots represent the similar type of data which is not relevant to show here.

In principle the last table contains all the information that one may need for the requested queries and for further data analysis. But, given the fact that not all the objects have observations in all the filters and there are preferred colours for certain analysis, I decided to create a new table with J-H and Y-J colours that will be used later on. (This will make some of the SQL queries easier and faster). For the "Colours" table the schema is shown in Table 3.

Column	**Type**	**SQL Type**	**Other**
ID_fitsfile	Float	INT	1-18
StarID	Integer	INT	Star identifier
FieldID	Integer	INT	1-3
Class_H	Float	INT	-1,0,1,2
Class_J	Float	INT	-1,0,1,2
Class_Y	Float	INT	-1,0,1,2
JH_colour	Float	DOUBLE	J-H
YJ_colour	Float	DOUBLE	Y-J

Table 3: Schema for the table "Colours". Note the Class identifiers which are -1 for stars, 0 for noise and +1 for non-stellar and -2 for boderline stellar.

With the shown schemas and the three tables ready we proceed to create the database which I call "Database_Juan.db"

1.b SQL Queries:

This part of the project consists on showing how the database meets the scientific needs of the survey manager who wants to have a database that can keep track of the location of the reduced images, the catalogues in each image and keep track of the colours of the stars and the variability in the Ks band. In order to achieve this, we perform SQL queries with certain special specifications in each case:

Q1: *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*

In order to make this query, we first define it as q.1 in the appendix and simply connect to it as seen in the jupyter notebook. The result of this query is displayed in Table 4.

File	MJD	Stars
Field-1-H	57257.044108	7982
Field-2-H	57258.044108	7725
Field-3-H	57258.044108	8022
Field-1-J	57257.0504323	7022
Field-2-J	57258.0504323	7354
Field-3-J	57258.0504323	7248
Field-1-Ks-E003	56829.0390512	7888
Field-1-Y	57267.1596647	6806
Field-2-Y	57268.1596647	7215
Field-3-Y	57268.1596647	7186
Field-1-Z	57267.1671072	6477
Field-2-Z	57268.1671072	6929
Field-3-Z	57268.1671072	6741

Table 4: Files that fulfill the conditions 56800>MJD>57300 with their corresponding MJD and the number of stars with S/N>5 in each image

Since the result of this query is less than 20 entries (13 files match the conditions) there is no need to represent it graphically and we can move on to the next query.

Q2: *Find the objects that have J-H > 1.5*

For this query we make use of the table "Colours" which already contains a column with the J-H colours of the colours that have measured magnitudes in those filters. The query is defined in q.2 in the appendix. We find that the number of objects that fulfill JH>1.5 is 7154 so we proceed to show the results graphically in a histogram (Figure 1).

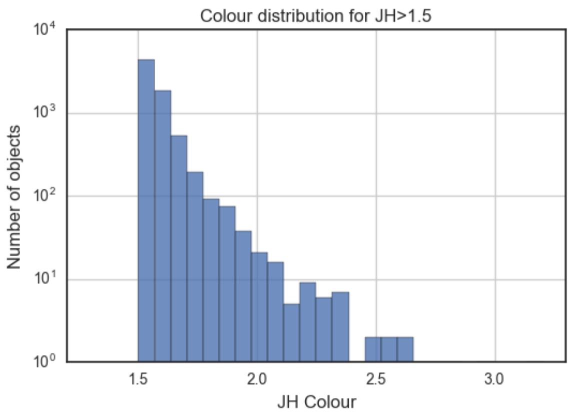

Figure 1: Distribution of J-H colour for the objects with J-H > 1.5 in the whole sample (in log scale for better visualization).

Note that in the second query we didn't need to specify whether the object was a star, noise, non-stellar or borderline stellar so we didn't need to include an additional filter in the object class. We can move on to the third query.

Q3: *Find the objects where Ks differs by more than 20 times the flux*

This query has an additional filter on the data. We first need to identify the images obtained with the Ks filter for each field and then we need to impose in those files the condition on the flux. There are several ways to interpret this query but I assumed that we wanted to retrieve the objects for which the flux (Flux1) in Ks differs from the mean flux (obtained as the average between the Ks magnitudes in all the observations for every case) by more than 20 times the flux uncertainty (dFlux1), that is:

$$\left| F(Ks_j) - \frac{1}{n}\sum_{i=1}^{n} F(Ks_i) \right| > 20 \times \Delta F(Ks_j) \quad (1)$$

Where F is the flux (used as Flux1 in every case), Ks_j indicates the catalogue number that has observations in the Ks band, ΔF is the uncertainty in the flux measured in the first aperture (dFlux1) and where n indicates the number of files with observations in Ks band for every individual field (thrice for field 1, once for field 2 and twice for field 3).

Since the observations in the Ks band are perfomed a different number of times for every filter I decided to make three separate queries (one for each field) in SQL which facilitates the finding of the objects and the calculation of the average between their fluxes for every catalogue.

Important note: I was tempted to select the objects that fulfil the condition per field and not per catalogue (that is, they fulfil the condition in all the catalogues in their respective field) but I assumed that this approach would make us lose some information such as in the case where an object has a small flux in one observation but good flux values in the other observations, in that case it could still be used for scientific purposes because it's still an object that fulfils the imposed condition even if it doesn't fulfil it in every observation.

The query for field 1 is shown in q.3 from the appendix (the other queries are all explicit in the noteboook) and the results for every catalogue are shown in Table 5.

Field	ID number	Stars
Field 1	4	6629
	5	6639
	6	7372
Field 2	11	6593
Field 3	16	6871
	17	6587

Table 5: Number of stars that fulfil Q3 per catalogue.

From the objects that fulfil the conditions let's see how many of them belong to each of the selected images (or catalogues) in Figure 2.

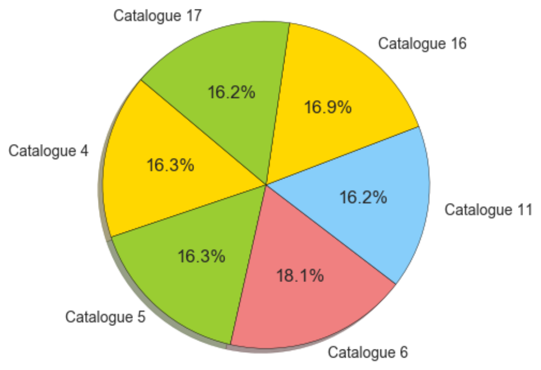

Figure 2: Percentages of objects that fulfill the conditions in query 3 for each catalogue (out of the total number of objects that fulfil Q3). Note that there will be many objects that belong to more than one catalogue because in each catalogue they fulfil the condition.

As we can see from the figure, there is a similar number of objects that fulfil the condition on the flux in each catalogue. We can argue that if there weren't many intrisic changes on the observed objects and if there wasn't too much variation in the observations performed more than once on the same field, the objects should in principle be the same which is a fair statement given the similarities on the amount of objects in each catalogue.

Q4: *Find all catalogues that exist for a given field*

This query is very straightforward and consists basically on selecting the images that belong to each field and showing their respective names, this is simply done in query q.4 from the appendix. The only difficulty in this query is that we must run it in a loop that goes on the number of fields that we have which is information we put in the query as a previously known value (in this case three fields). The results of the query are shown in Table 6.

Field	Name file	Name .fits
Field 1	Field-1-Z	Z-ADP.2017-01-18T11:58:36.905.fits
	Field-1-J	J-ADP.2017-01-18T11:58:35.781.fits
	Field-1-H	H-ADP.2017-01-18T11:58:35.780.fits
	Field-1-Ks-E002	Ks-ADP.2016-05-25T15:33:39.546.fits
	Field-1-Ks-E001	Ks-ADP.2017-01-18T11:58:39.907.fits
	Field-1-Ks-E003	Ks-ADP.2016-05-25T15:33:43.377.fits
	Field-1-Y	Y-ADP.2017-01-18T11:58:36.901.fits
Field 2	Field-2-Z	Z-ADP.2017-01-18T11:58:36.905b.fits
	Field-2-J	J-ADP.2017-01-18T11:58:35.781b.fits
	Field-2-H	H-ADP.2017-01-18T11:58:35.780b.fits
	Field-2-Ks-E001	Ks-ADP.2016-05-25T15:33:39.546b.fits
	Field-2-Y	Y-ADP.2017-01-18T11:58:36.901b.fits
Field 3	Field-3-Z	Z-ADP.2017-01-18T11:58:36.905c.fits
	Field-3-J	J-ADP.2017-01-18T11:58:35.781c.fits
	Field-3-H	H-ADP.2017-01-18T11:58:35.780c.fits
	Field-3-Ks-E002	Ks-ADP.2016-05-25T15:33:39.546c.fits
	Field-3-Ks-E001	Ks-ADP.2017-01-18T11:58:39.907c.fits
	Field-3-Y	Y-ADP.2017-01-18T11:58:36.901c.fits

Table 6: Images with their short and long names for every field.

The number of images is 18 so we don't display them graphically. Let's move on to the last query.

Q5: *For a given image 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*

This query has one ambiguity related to the nature of the survey. Since there are three fields and each of them has observations in the same filters, then one must specify what field the studied image belongs to. This way, one can study each image individually and show the requested statistics associated to it (in this case the stars that fulfill the requirement of the signal to noise ratio). Once this was understood I decided to write my code as a function of the chosen field, in my case I define a variable called `fieldID` which is the input that one must change to study a different field. By doing this one can find the required information for the three fields as displayed in Table 7. The query is written in q.5 from the appendix.

Observed Field	Filters	Number of Stars
Field 1	Y	4218
	Z	3740
	J	4829
	H	6356
	Ks4	4616
	Ks5	4196
	Ks6	6574
Field 2	Y	4998
	Z	4614
	J	5336
	H	6186
	Ks11	4156
Field 3	Y	4807
	Z	4271
	J	5104
	H	6311
	Ks16	4871
	Ks17	4110

Table 7: Number of stars that fulfil the requirement in each file for each of the three observed fields.

Since in all the cases the results are very large numbers, I present graphically the results for

Field 1 since it is the most representative of the three fields (it has three images in the Ks filter). The results of this field are displayed in Figure 3.

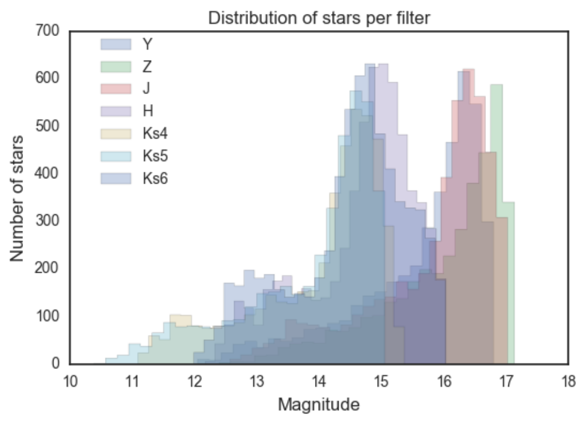

Figure 3: Distribution of stars per filter in field 1. This histogram shows that there is a similar amount of stars in all the filters and there are two peaks so it's useful to make individual histograms.

Since we can't really see the distribution of the data in this histogram, I decided to make individual histograms for all the filters in the field as shown in Figure 4.

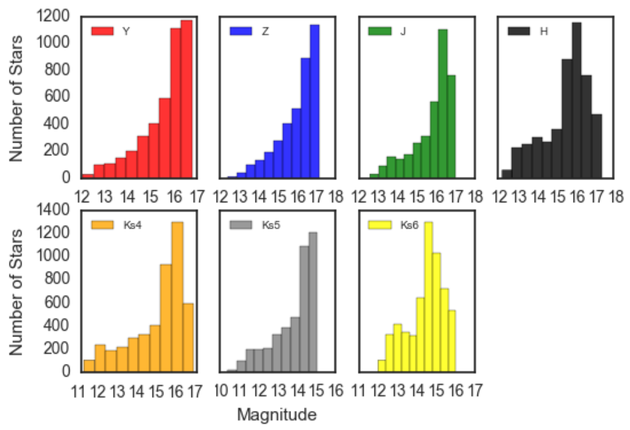

Figure 4: Distribution of stars that fulfil condition of query 5 for each individual filter. Note that the peak in the distributions of the Ks magnitudes is located at smaller magnitudes than the peak of the other filters.

For practical purposes I don't include here the figures produced for field 2 and 3 in this query, but the user can simply change the variable fieldID to any of those numbers in the jupyter notebook and get immediately the plots and the numbers for those fields. This shows that the built database can fulfill the scientific requirements of the survey manager and so it can be used for scientific purposes.

1.c Simulated stars for Euclid

—>Follow the jupyter notebook<—
`YJ_JH_distribution_func.ipynb`

This part of the project has to do with an application of the learned topics in the course to Astronomy. We are asked to use the distribution of the database to create 100,000 simulated stars for the Euclid space mission which will have the filters Y, J and H.

First we must connect once again to our database and in my case, use the information contained in the table "Colours". The query to get the data that we will connect to a pandas dataframe is q.6. Note that since there are some "-nan-" values in our data, then we need to put an additional filter in our queries to avoid them, in this case the error values are in the JH_colour column so we need to include `...JH_colour IS NOT NULL...` in the query.

After connecting this data to pandas we first plot the real stars in the JH - YJ plane as shown in Figure 5 to see what the real distribution is.

Now, for simulating our sample of stars we will use two different methods of mixture models from sklearn on our data. These are probabilistic models that allow us to create or represent subpopulations within an overall population which is our input. In other words, they are mixture models that represent the probability distribution of observations on the real data.

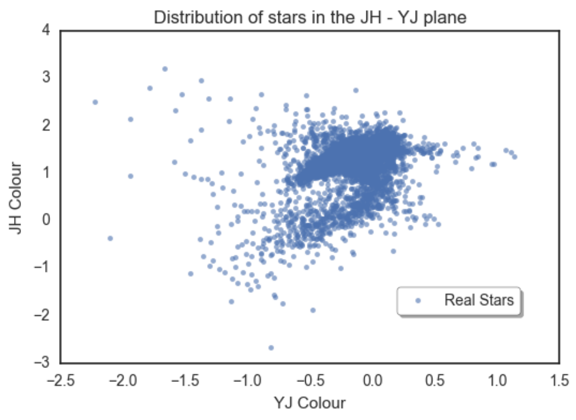

Figure 5: Distribution of the real stars from the database on the JH - YJ plane. We expect the simulated stars to follow a similar distribution.

The first model I use is the GaussianMiture model which is a model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions which is the number we must fit accordingly. The best number of Gaussians (10 in this case) was estimated with a visual inspection but in the second method it was properly estimated. The results for the Gaussian mixture model is shown in Figure 6.

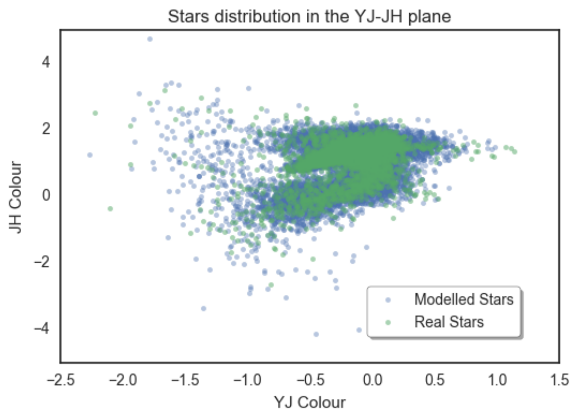

Figure 6: Modelled stars (blue) and the real stars (green) on top for the simple Gaussian-Misxture model where the amount of Gaussians was chosen to be 10.

We can also use a more robust method which is the BayesianGaussianMixture model, a variant of the Gaussian mixture model that implements variational inference algorithms. This method requires some extra parametrizations for the variational inference of the model.

The main reason to use this method is that it allows me to estimate the number of gaussians that best fit the data (and not just make a guess like in the previous method). This is done using the prediction attribute of BayessianGaussianMixture after training the fit. I chose `max_iter = 5000` for the number of EM (expectation maximization) iterations to perform and an upper bound of 30 gaussians in the fitting. The best fit of the model was found using 16 gaussians. The simulated stars look very similar to the first approach done using simple GaussianMixture and we compare them in Figure 7.

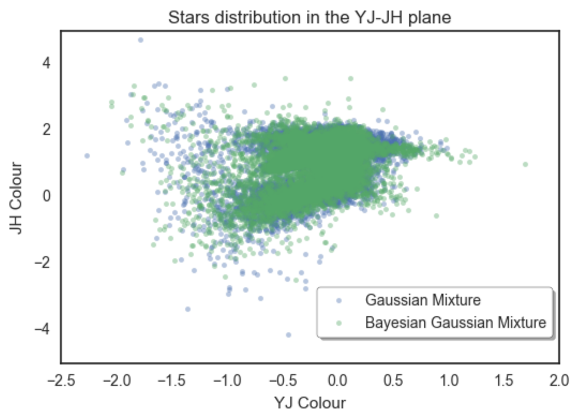

Figure 7: Comparison between the two methods used to find the distribution function of stars for Euclid. Gaussian Mixture method (blue) and Bayessian Gaussian Mixture (green) on top.

In Figure 7 we can see that the two methods reproduce fairly well the data although the Bayessian Gaussian mixture model is able to reproduce additional features on the data such as some horizontal branches which have a low density and thus are easier to fit with an optimized number number of Gaussians which in this case turns out to be 16.

Problem 2. PHOTOMETRIC REDSHIFT OF GALAXIES

—>Follow the jupyter notebook<—
`Photo_z_linear_regression.ipynb`

Using multi band photometry data to get redshifts (photometric redshifts) is not only a useful method to get redshifts of fainter objects than accessible by spectroscopy, but also because the efficiency in terms of the number of objects with redshift estimates per unit time is largely increased.

A lot of research is being done with the purpose of improving the determination of distances of galaxies using photometric redshifts. Given the advantage that this technique provides in terms of observing time, and the possibiliy of observing big regions of the sky at the same time, this technique will be used in some future cosmological surveys that require a very significant improvement of the determination of redshifts through photometry, more specifically they should fulfil:

$$\langle |(z_{phot} - z_{spec}) \,/\, (1 + z_{spec})| \rangle < 0.01 \tag{2}$$

In order to achieve this we must construct accurate models that predict photometric redshifts from magnitudes that reproduce very well the spectroscopic redshifts that we trust so much. We can then quantify the discrepancy between photometric and spectroscopic redshifts as:

$$E(\theta) = median\left(|(z_{spec} - f(\theta)) \,/\, (1 + z_{spec})|\right) \tag{3}$$

Where $f(\theta)$ is the photometric redshift found with the chosen method.

In this part of the project we are asked to get some photometric redshifts from galaxies using regression estimators on the data and try to minimize the error associated to its implementation. For this purpose we have two tables (named A for PhotoZFileA.vot and B for PhotoZFileB.vot) which contain the magnitude in r and the $u-g$, $g-r$, $r-i$ and $i-z$ colours from which we can easily compute the respective magnitudes in each filter.

2.a Feature extraction

Feature extraction is a powerful technique used in data mining that allows for the simplification and optimization on the processing of lots of data and facilitates the identification of the main characteristics and behavior of a certain sample. It consists on identifying redundant features and general trends on the data in order to build a simpler version of it which contains useful information about the original sample but which has the characteristic of being non-redundant.

One of the great advantages of using feature extraction is that it reduces computational time and allows for the interpretation of the data when it's properly applied on the sample and when it's able to describe the data with enough accuracy. The goal of well performed feature extraction is to get an output that contains all the relevant information from the initial input, so that it is possible to perform desired tasks on the reduced dataset rather than in the big initial dataset.

When performing feature extraction it is useful to first perform an heuristic approach which is simply a superficial understanding of the data through plots or getting basic statistical tests to get an idea on how the data is layed out. It's also important to consider a single feature ranking approach in which features are ranked by relevance and which will allow for a consistent and coherent feature extraction. One must also consider the performance bounds and the computational resources available for getting accurate results, taking into account that there must be a point where time and performance optimization can be both achieved.

2.b Linear regression methods

Here I design a regression estimator to predict photometric redshifts from the data using ridge, Lasso and linear regression. The results suggest that linear regression is a faster and better method because it gives a smaller error and is good enough for this set of data.

There are certain ways to calculate photometric redshifts from magnitudes and colours, in this case I will use linear combinations of the magnitudes (as discussed in [4]) to get z_{phot} so that the regression estimators can predict the coefficients (a, b, c, d, e, f) that best reproduce the given spectroscopic redshift values. Equation [4] shows how the photo-z values are calculated as a linear combination of only magnitudes (M_r, M_i, Mz, M_g, M_u).

$$z_{phot} = a + bM_r + cM_i + dM_z + eM_g + fM_u \quad (4)$$

The values of the coefficients for each regressor are saved and shown in the jupyter notebook and not displayed here since it it more insighful to focus on the training error associated to each method. This error is quantified by using the spectroscopic data as the real data and comparing it to the fitted funcions as shown in eq [3]. The found values in each case are shown in Table 8.

Method	α	**Training error**
Ridge	0.0001	0.0146097
LASSO	0.000001	0.0155897
Linear regression	0	0.0145589

Table 8: Training errors found for the Ridge, Lasso and Linear regression methods. Note that the best values of the regularization strength (α) are very small for Lasso and Ridge regression which approach linear regression where $\alpha = 0$.

The values of the training error for all the methods are very similar and around 0.0146, and I decided to take the linear regression method as my preferred method given the fact that the other ones only approach to it with the variation of the regularization strength constant when I try to decrease the training error. Figure 8 shows the results of the photo-z found for each method.

2.c Generalization error

The error that we estimated on the training sample is not a reliable estimate of the generalization error of the estimator because the training error is the error of a trained model on the training set used to generate it, that is, the error that comes out after we've obtained the best fit on that specific data set. This implies that the error is directly correlated to the data from where it was obtained thus creating a bias on its estimation.

On the other hand, the generalization error of an estimator should be obtained as the error of a trained model on the whole space of possible data to get beyond that bias. In other words, it should be obtained from a model which has no association whatsoever to the data on which it is being computed upon.

To quantify the generalization error of the estimator derived in problem b we use the three methods (although the preferred one is linear regression). This is done by using the model we created in part b (the model that best recreates the spectroscopic redshifts) and applying this model to the set of data B. This will basically calculate the generalizaed error of the estimator and we define it again as eq [3]. The errors are shown in table 9

Method	**Training error**	**Generalized error**
Ridge	0.0146096803208	0.0146471352295
LASSO	0.0155897203138	0.0155995565379
LR	0.0145588611233	0.0146014572286

Table 9: Training and generalized errors for Ridge, Lasso and Linear Regression.

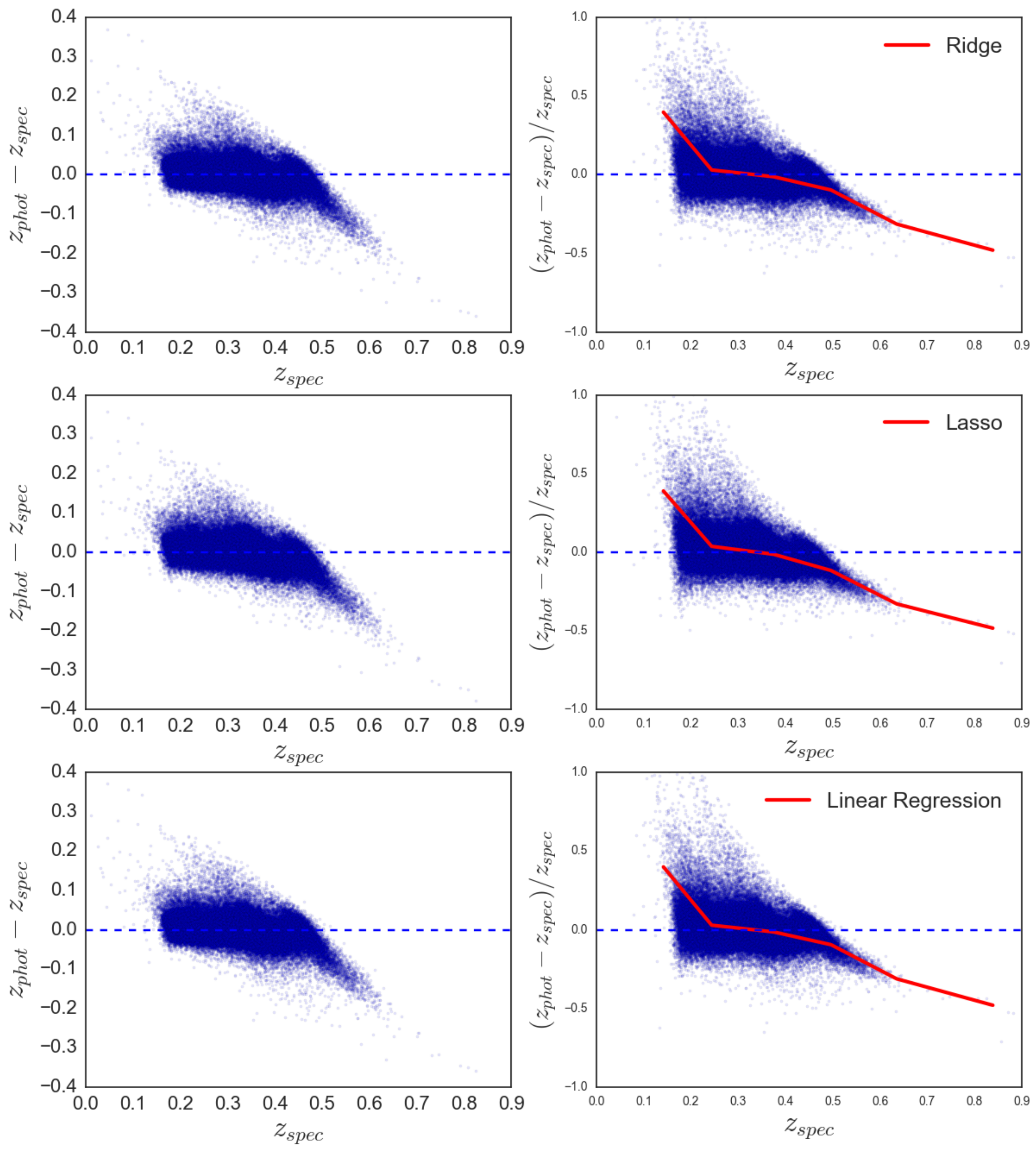

Figure 8: Left panel: Difference between the photo-z and the spectroscopic redshifts for each of the implemented methods. Right panel: Residuals of the three methods and the median trend line calculated in bins (for Ridge, Lasso and Linear Regression respectively).

2.d Photo-z estimator using K-nearest neighbours

—>Follow the jupyter notebook<—
`Regression_KNN.ipynb`

In this part we are asked to implement a photo-z estimator using different regression methods discussed in the course. I decided to use K-nearest neighbours because it is a very efficient approach in terms of computing and already retrieves a smaller error than the generalized error of the linear regressor methods implemented before.

KNN is used for regression and classification and it is a simple algorithm that stores all available cases and predict the numerical target based on a similarity measure (e.g., distance functions). It is a non-parametric technique and has been used since the 1970's as mentioned in [5].

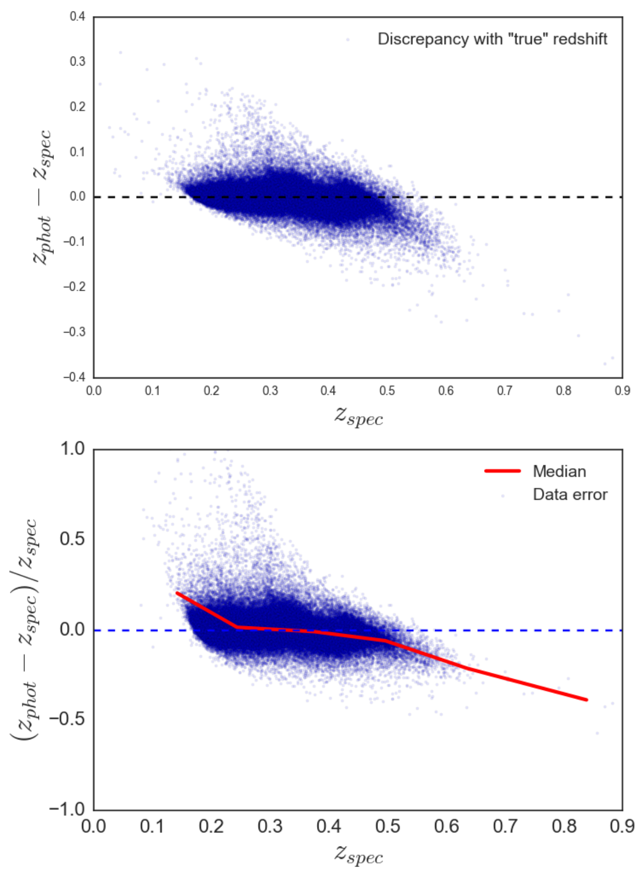

Figure 9: Top: Discrepancy of the estimated data with the true redshift. Botton: Residuals of this method and the median trend line.

The algorithm applied here consists of calculating the average of the numerical target of the K nearest neighbors (which I set to 5 through a bias-variance tradeoff analysis). KNN regression uses the same distance functions as KNN classification which can be Euclidean, Manhattan or minkowsky, in my case I decided to let the parameter of the metric be the default which is Euclidean. The output is the property value for each object which is the average of the values of its k nearest neighbors.

Implementing this method on the data I get a training error of $E(\theta)_{train}$= 0.0107609493248. Adittionally, using this new model on data B to obtain the generalized error I get a generalization error of $E(\theta)_{gen}$ = 0.0133751684158 so we see that there is an improvement since this error is smaller than the one obtained in all the linear regression models.

We can see clearly in Figure 9 that the discrepancy between the computed photometric redshifts and the real redshifts is small than it was with the linear regression methods used before. The most distinctive feature on the plot is the first part of the median trend line (first bin) which has a smaller slope than the ones obtained with the linear regression methods.

2.e Discussion and comparison between KNN and linear regression methods

First of all it is important to recall that linear regression methods are parametric approaches whereas KNN is a non-parametric approach. Some of the advantages of the parametric approach is that it's easy to fit and it fits a small number of parameters but it also has disadvantages such as the fact that it makes strong assumptions about the form of the function that it's fitting which might deviate significantly from the true model.

In this specific science case, we still don't know the true model to get photometric redshifts from magnitudes and colours so using linear regression imposes a bias on the results and might make predictions which are not very accurate.

On the other hand, a non-parametric approach such as k-nearest neighbours doesn't assume an explicit form of the fitting function so that it provides a more flexible approach and could in principle obtain a more realistic set of data.

Even though it is based on one of the most simple algorithms for regression, it gives accurate results if the training data is large such as in our case.

Some of the disadvantages of KNN method is that it doesn't learn from the training data and it can be more complex to understand and interpret because it's implementation is not as intuitive as the linear methods are. It also can rely on the use of a large number of coefficients even if the true model doesn't need them. Additionally, in contrast to linear regression methods (where the implementation is straightforward), in KNN one must find an appropriate value for the numbers of neighbours to use by doing a very comprehensive analysis of the bias-variances trade-off.

In our case, the decrease in the generalization error is very small to the one obtained with the linear regression methods so it's arguable that in this specific problem a linear regression method can be enough to get the photometric redshifts even more when some authors in the literature argue that the function to obtain the photo-z is in fact a linear function of the magnitudes.

—>The codes were written using the language, packages and libraries Python 2.7, Sklearn, SQL, Matplotlib, Pandas, Seaborn, Numpy and the report was fully written on LaTeX<—

References

[1] https://github.com/jbrinchmann/DDM2017.git

[2] http://scikit-learn.org/stable/modules/mixture.html

[3] "*An introduction to feature extraction*" Isabelle Guyon and André Elisseeff, ClopiNet, Berkeley, CA 94708, USA.

[4] Connolly et al. (1995a)

[5] http://www.saedsayad.com/k_nearest_neighbors_reg.htm

[6] "*The elements of Statistical Learning*", Scott Fortmann.

Appendix

First SQL query:

```
Q1_que = """SELECT filename, MJD, numstars FROM (SELECT inf.name_file as filename,
inf.MJD as MJD, COUNT(data.StarID) as numstars FROM info as inf, All_data as data
WHERE data.Class=-1.0 AND data.ID_fitsfile == inf.ID AND data.SN > 5 GROUP BY inf.Filena
WHERE MJD>56800 AND MJD<57300"""
```

Second SQL query:

```
Q2_que = """SELECT Colours.JH_colour From Colours WHERE Colours.JH_colour > 1.5"""
```

Third SQL query (in this case only for field 1):

```
Q3_que_1 = """SELECT inf.ID as ident, inf.Filter as fil, inf.FieldID as field, data.Flux
as flux, data.StarID as starid FROM info as inf, All_data as data WHERE field="1"
AND data.Filter="Ks" and inf.ID == data.ID_fitsfile AND ABS(data.Flux1 - ((SELECT
AVG(data.Flux1) FROM All_data as data WHERE data.Filter="Ks" AND data.StarID ==
data.StarID GROUP BY data.StarID))) > ABS(20*data.dFlux1)"""
```

Fourth SQL query:

```
Q4_que = """SELECT fieldid, file FROM (SELECT inf.name_file as file, inf.FieldID
as fieldid FROM info as inf) WHERE fieldid == """.format(i)
```

Fifth SQL query:

```
Q5_que_filter = """SELECT inf.ID as ident, inf.Filter as fil, data.Mag1 as mag,
data.StarID as starid FROM info as inf, All_data as data WHERE inf.FieldID ==  AND
data.Class=-1 AND inf.ID == data.ID_fitsfile AND data.SN > 30""".format(fieldID)
```

Simulation stars SQL query:

```
query = """SELECT JH_colour, YJ_colour FROM Colours WHERE Colours.Class_H==-1.0 AND
JH_colour IS NOT NULL"""
```