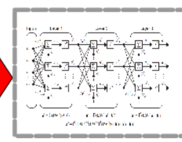
The Goal: Predicting photometric redshift using photometric data (as the input) and spectroscopic redshift (as the target). Here the model we use is a SVM.

u-b	b-v	v-r	r-i	i-z	z-j	j-k
0.415	0.938	0.456	0.213	0.14	0.121	0.275
0.44	0.985	0.455	0.271	0.249	0.163	0.476
-0.157	0.072	0.143	0.43	0.451	0.224	0.03
-0.167	0.488	0.723	0.039	-0.031	0.343	-0.044
-0.618	0.786	0.864	0.846	0.861	0.496	0.628
0.043	0.207	0.184	0.425	-0.256	-0.704	0.82
0.356	0.809	0.44	0.195	0.124	0.207	0.098
1.167	1.259	0.694	0.335	0.595	0.364	0.117
-0.334	1.994	1.454	0.539	0.711	0.564	0.53
0.147	0.719	0.249	0.038	-0.028	-0.115	0.041
1.869	1.098	0.978	0.316	0.52	0.444	0.419
0.442	0.592	0.824	0.226	0.305	0.379	0.174
-0.461	0.295	0.574	0.301	-0.334	0.426	-0.137
0.003	0.618	0.741	0.183	0.376	0.241	0.269
0.057	1.202	1.164	0.393	0.471	0.454	0.365
0.831	0.616	0.919	0.369	0.433	0.269	0.355
0.286	1.083	1.068	0.38	0.606	0.362	0.509
-0.149	2.264	1.513	0.841	0.632	0.445	0.341
0.359	0.872	0.922	0.3	0.487	0.319	0.455
0.585	0.044	0.576	0.186	-0.322	0.151	0.348
0.291	0.342	0.556	0.209	0.161	0.026	0.122
0.463	1.431	1.06	0.348	0.547	0.294	0.378
-0.1	0.784	0.741	0.207	0.307	0.325	0.251
1.01	1.714	0.851	0.312	0.372	0.3	0.278
0.573	0.294	0.475	0.587	0.594	0.332	0.368
0.202	0.771	0.936	0.683	0.679	0.509	0.466
0.849	1.155	0.565	0.244	0.291	0.209	0.162
0.46	0.812	0.95	0.721	0.639	0.396	0.595
0.294	0.651	0.66	0.154	0.24	0.29	0.186
1.847	1.422	1.139	0.449	0.588	0.572	0.395

A model



Redshift (z)
0.3588
0.3592
1.3810
0.6710
1.2220
0.7370
0.6250
0.1239
0.7350
0.2032
0.6830
0.7338
0.8321
0.7612
0.7340
0.9055
0.9906
0.9641
0.7619
0.9809
0.8410
0.6690
0.6691
0.4147
1.3288
1.0150
0.2460
1.0973
0.6278
0.6692

Input [713x7]

Target (713x1)

From GOODS-MUSIC catalog Santini et al. (2009)

Loading packages

```
In [1]: %matplotlib inline
import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from pandas.plotting import scatter_matrix
from sklearn.neighbors import KNeighborsRegressor
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.model_selection import train_test_split
from sklearn import linear_model
import sys
from scipy.interpolate import interp1d
from matplotlib import pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
print('Done!')
```

```
Done!
```

```
In [2]: X=np.load('inp_redshift.npy')  # Load the input
Y=np.load('tar_redshift.npy')  # Load the target

print ('Done!')
```

```
Done!
```

```
In [3]: print(np.shape(X),np.shape(Y)) #Check the size and dimension of the input and
```

```
(713, 7) (713,)
```

```
In [4]: Y = np.reshape(Y, (-1, 1))
```

```
In [5]: print(np.shape(X),np.shape(Y))
```

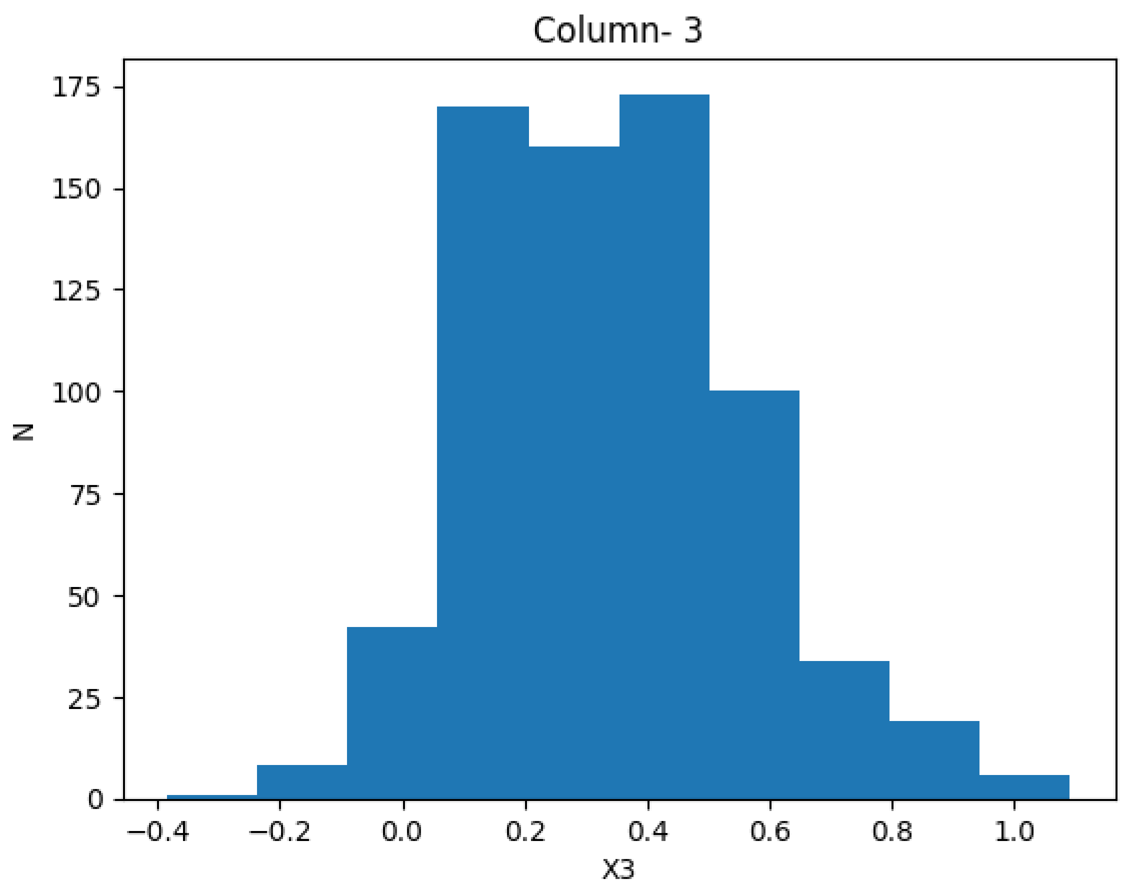
```
(713, 7) (713, 1)
```

Exploring the data and seeing the distribution of a selected column

In [ ]:

In [6]:
```python
# Choose a column to see the distribution

n_column =3

plt.hist(X[:,n_column])
plt.title('Column- '+str(n_column))
plt.ylabel('N')
plt.xlabel("X"+str(n_column))
```

Out[6]: Text(0.5, 0, 'X3')



In [ ]:

Randomly separate 713 samples to the training set (75%) and validation set (25%). The corresponding targets are also separated.

In [7]:
```
from sklearn.model_selection import train_test_split

X_tr,X_va,Y_tr, Y_va = train_test_split(X,Y ,test_size=0.25 )

print ('training set == ',np.shape(X_tr),np.shape(Y_tr),',, validation set ==
```

```
training set ==  (534, 7) (534, 1) ,, validation set ==  (179, 7) (179, 1)
```

Normalization.

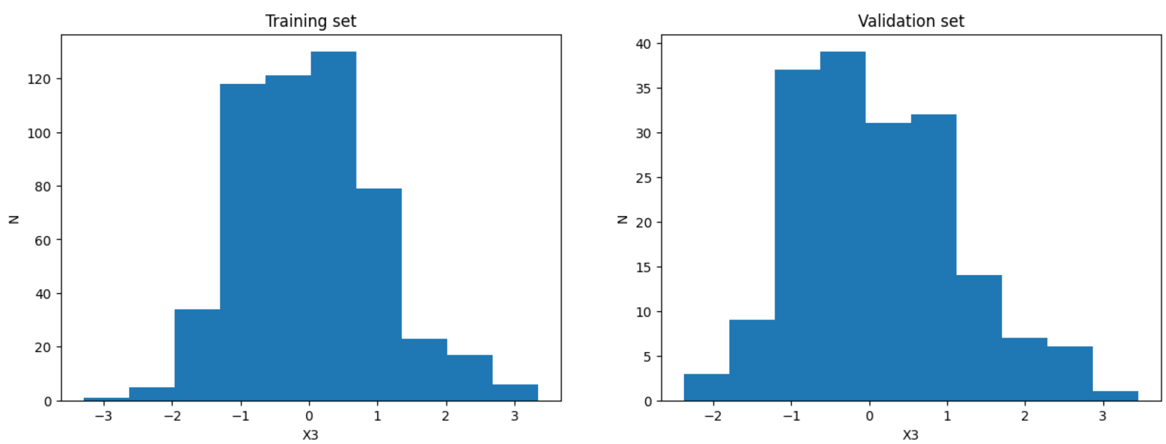
In [8]:
```
#Line #1: Import a model, for normalization, like StandardScaler (https://sci

#Line #2: fitting (finding the parameters of the model based on the training

#Line #3: Predicted (transformed) values for the training set

#Line #4: Predicted (transformed) values for the validation set (using the mo



scaler_S= StandardScaler().fit(X_tr)  # line #2
X_tr_Norm= scaler_S.transform(X_tr) # line # 3

X_va_Norm= scaler_S.transform(X_va)  # Line #4

print('Done!')
```

```
Done!
```

Comparing the distributions from the nomalized training and validation sets

In [9]:
```
n_column = 3

fig = plt.figure(figsize=(15, 5))

plt.subplot(1, 2, 1)
plt.hist(X_tr_Norm[:,n_column])
plt.title('Training set')
plt.ylabel('N')
plt.xlabel("X"+str(n_column))


plt.subplot(1, 2, 2)
plt.hist(X_va_Norm[:,n_column])
plt.title('Validation set')
plt.ylabel('N')
plt.xlabel("X"+str(n_column))
```

Out[9]:
```
Text(0.5, 0, 'X3')
```



In [10]:
```
# Change the shape of the target (if you have a one-component  target )
# Y=np.reshape(Y,-1)
# print(np.shape(X),np.shape(Y))
```

In [11]:
```
from sklearn.model_selection import GridSearchCV


param_grid = [{"n_neighbors":[1,3,5,8,10,12,15,20,100, 200],"p":[1,2]}]  # ch

reg= KNeighborsRegressor()

grid_search = GridSearchCV(reg, param_grid, cv=5)

grid_search.fit(X_tr_Norm,Y_tr)


print (grid_search.best_params_)
```

```
{'n_neighbors': 12, 'p': 1}
```

In [12]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler


knn_pip = Pipeline([("scaler", StandardScaler()),
 ("knn", KNeighborsRegressor(n_neighbors=25, p=2 )),])

knn_pip.fit(X_tr,Y_tr)

Y_pre_va=knn_pip.predict(X_va)
Y_pre_tr=knn_pip.predict(X_tr)

np.shape(Y_pre_va)
```

Out[12]:
```
(179, 1)
```

In [ ]:
```

```

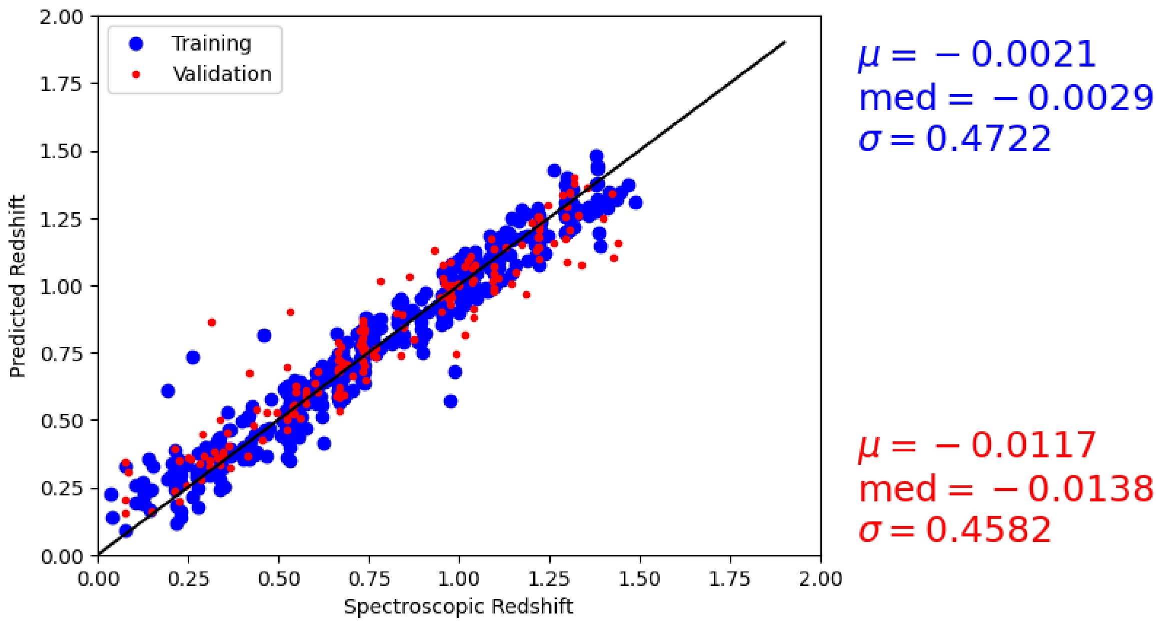
For more information:

Type *Markdown* and LaTeX: α^2

In [13]:
```
from sklearn.svm import SVR

reg= SVR( kernel='rbf', degree=3, tol=0.001, C=1)


reg.fit (X_tr_Norm,Y_tr)  # fit the model with training set

#'predictions for training and validation sets'
Y_tr_pred= reg.predict(X_tr_Norm)
Y_va_pred= reg.predict(X_va_Norm)


plt.figure(3)
plt.plot(Y_tr,Y_tr_pred,'ob')
plt.plot(Y_va,Y_va_pred,'.r')

plt.plot(np.arange(0,2,.1),  np.arange(0,2,.1),'-k')
plt.xlabel('Spectroscopic Redshift')
plt.ylabel('Predicted Redshift')
plt.legend(['Training', 'Validation'])
plt.xlim([0,2])
plt.ylim([0,2])

#Statistical information regarding training and validation predictions
mu = np.mean(Y_tr-Y_tr_pred)
median = np.median(Y_tr-Y_tr_pred)
sigma = np.std(Y_tr-Y_tr_pred)

muv = np.mean(Y_va-Y_va_pred)
medianv = np.median(Y_va-Y_va_pred)
sigmav = np.std(Y_va-Y_va_pred)

textstr = '$\mu=%.4f$\n$\mathrm{med}=%.4f$\n$\sigma=%.4f$'%(mu, median, sigma
textstrv = '$\mu=%.4f$\n$\mathrm{med}=%.4f$\n$\sigma=%.4f$'%(muv, medianv, si


plt.text(2.1,1.5,textstr, color='b',fontsize=18)
plt.text(2.1,.05,textstrv, color='r',fontsize=18)
```

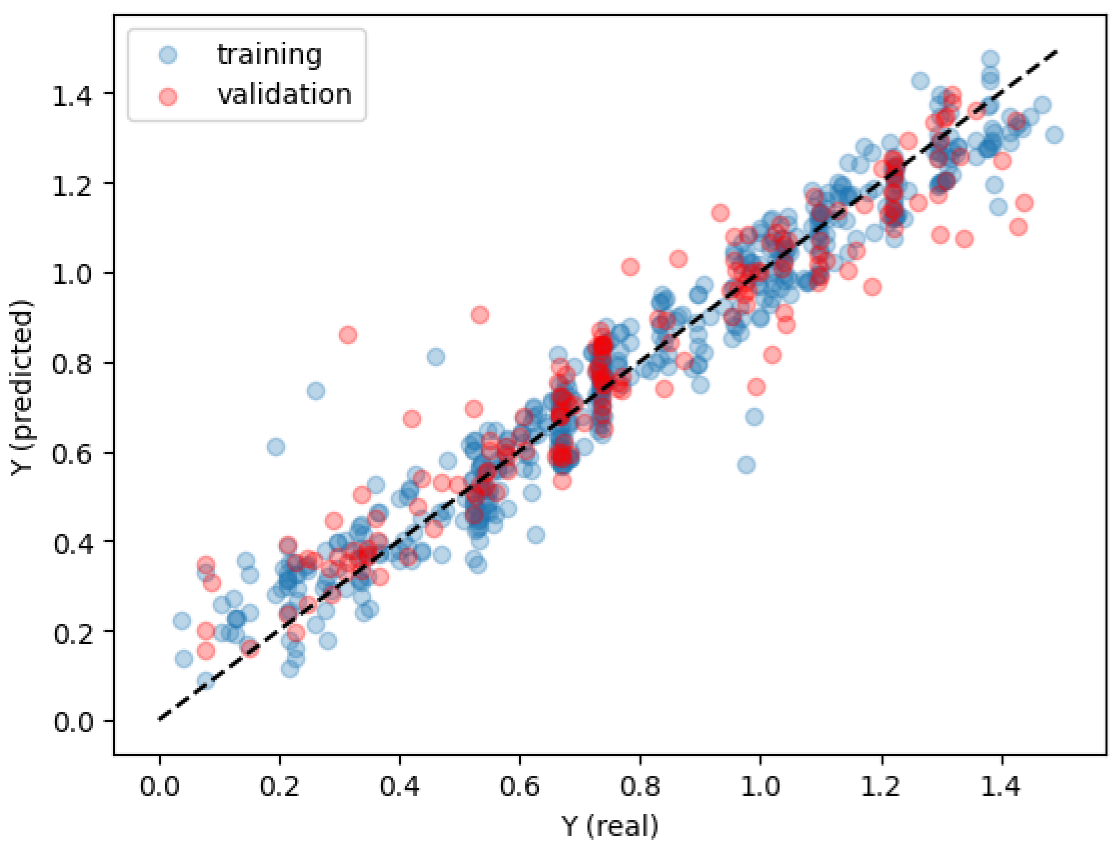
```
C:\Users\hteim\anaconda3\envs\tf\lib\site-packages\sklearn\utils\validation.
py:1111: DataConversionWarning: A column-vector y was passed when a 1d array
was expected. Please change the shape of y to (n_samples, ), for example usi
ng ravel().
  y = column_or_1d(y, warn=True)
```

Out[13]:
```
Text(2.1, 0.05, '$\\mu=-0.0117$\n$\\mathrm{med}=-0.0138$\n$\\sigma=0.4582$')
```



In [14]:
```
# The same as above, with different visualization
plt.scatter(Y_tr,Y_tr_pred,label='training',alpha=.3)
plt.scatter(Y_va,Y_va_pred,label='validation',color='r',alpha=.3)
plt.xlabel('Y (real)')
plt.ylabel('Y (predicted)')
plt.plot([0,1.5],[0,1.5],'--k')
plt.legend()
```

Out[14]: <matplotlib.legend.Legend at 0x24e0ef62f40>



In [ ]:

```

```

In [15]:

```
Y_tr=np.reshape(Y_tr,(-1,1))
Y_va=np.reshape(Y_va,(-1,1))
Y_tr_pred=np.reshape(Y_tr_pred,(-1,1))
Y_va_pred=np.reshape(Y_va_pred,(-1,1))
```

In [16]:

```
np.shape(Y_tr_pred)
```

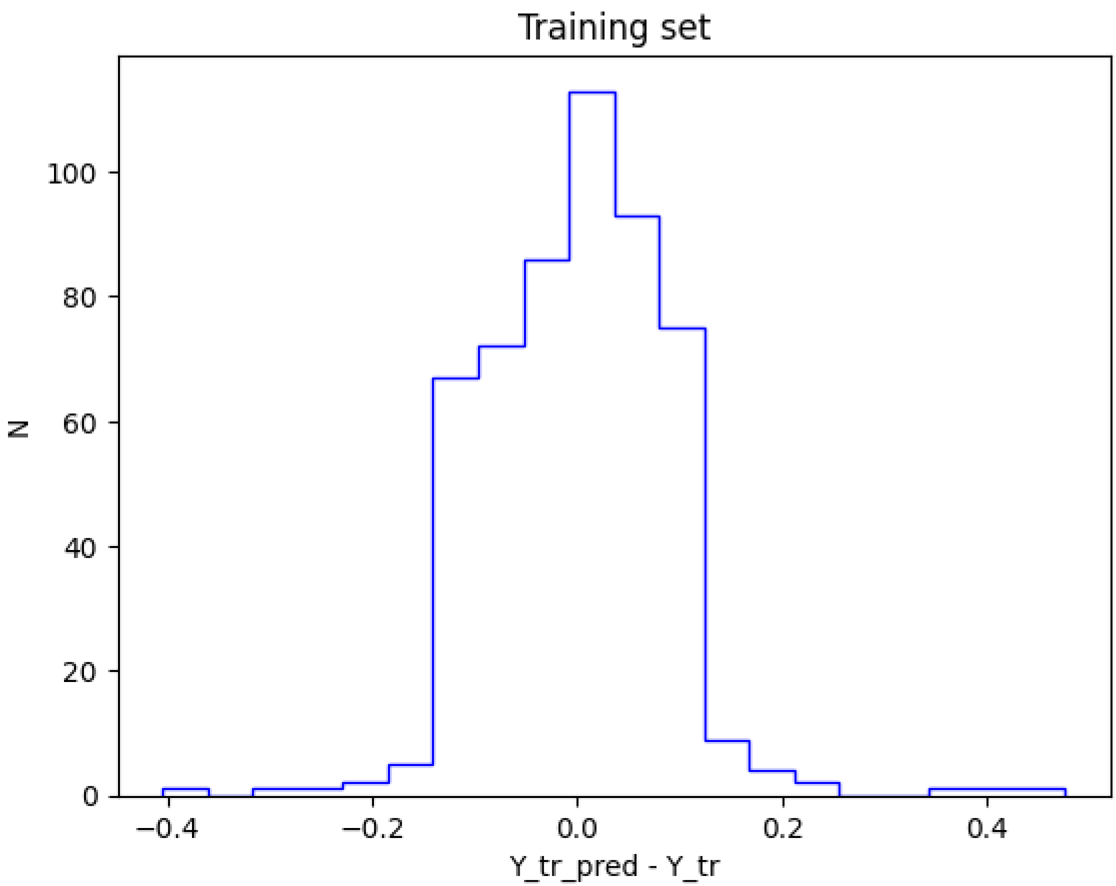
Out[16]:

```
(534, 1)
```

Comparing predicted and actual values

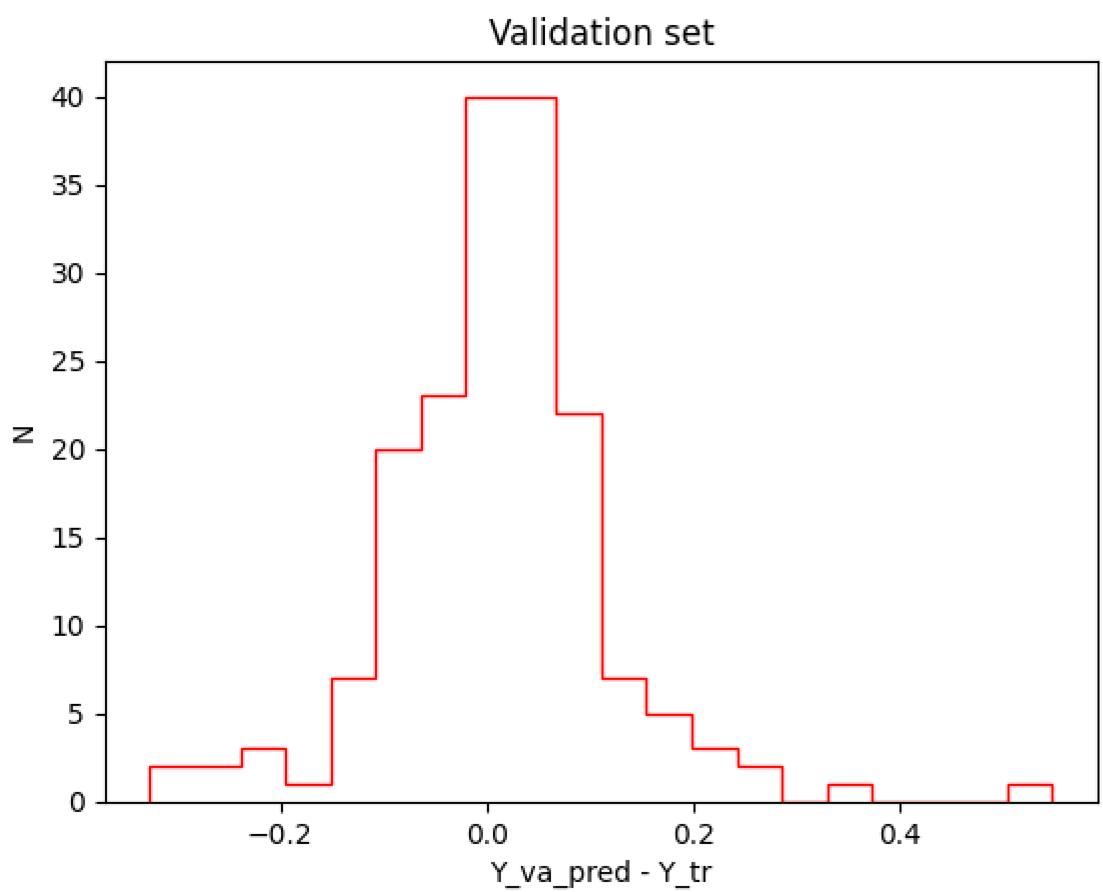
In [17]:
```
# Inspect the distribution of the difference between the predicted and actual
plt.hist(Y_tr_pred-Y_tr,20,color='b',histtype='step')
plt.xlabel('Y_tr_pred - Y_tr')
plt.ylabel('N')
plt.title('Training set')
print ('mean = ',np.mean(Y_tr_pred-Y_tr) )
print ('median = ',np.median(Y_tr_pred-Y_tr) )
print ('SD = ',np.std(Y_tr_pred-Y_tr) )
```

```
mean =  0.0020887055153040184
median =  0.003916985684148222
SD =  0.08307781057318774
```



In [18]:
```
# Inspect the distribution of the difference between the predicted and actual
plt.hist(Y_va_pred-Y_va,20,color='r',histtype='step')
plt.xlabel('Y_va_pred - Y_tr')
plt.ylabel('N')
plt.title('Validation set')

print ('mean = ',np.mean(Y_va_pred-Y_va) )
print ('median = ',np.median(Y_va_pred-Y_va) )
print ('SD = ',np.std(Y_va_pred-Y_va) )
```

```
mean =  0.011705293834323905
median =  0.012931797151971014
SD =  0.10514769314740746
```

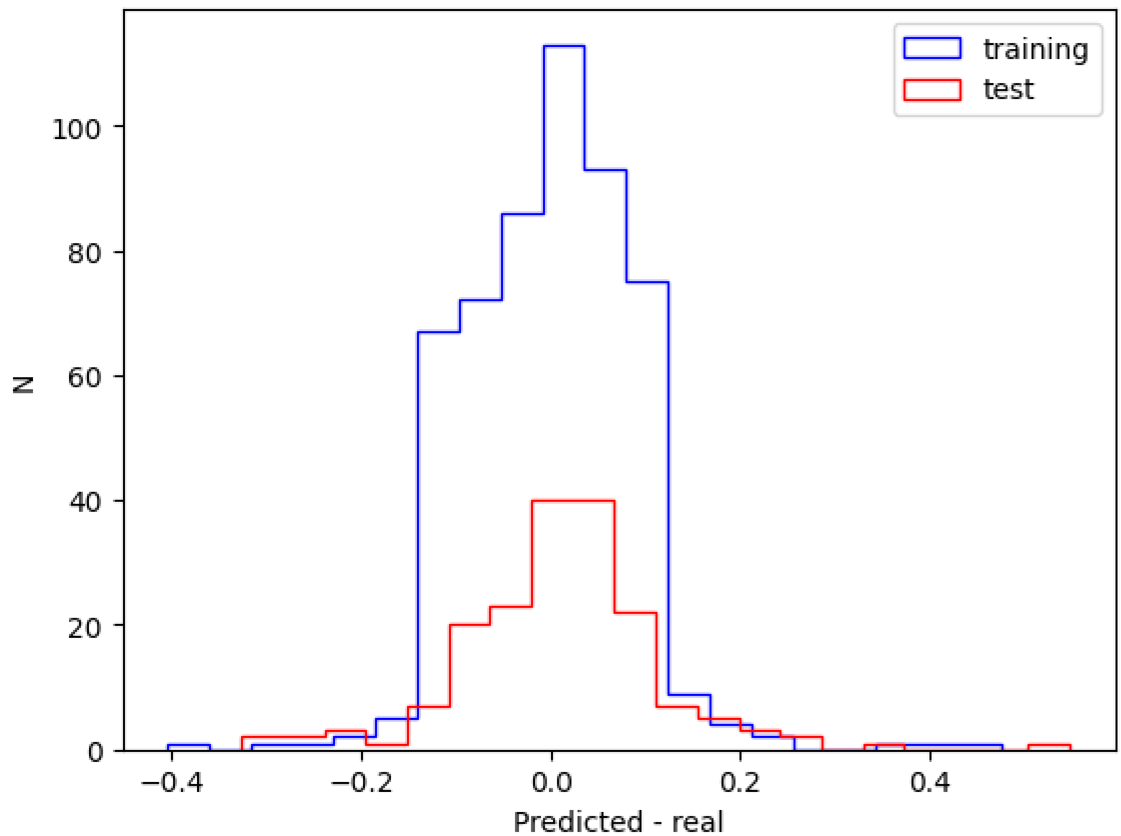


In [19]:
```
np.shape(Y_va_pred)
```

Out[19]:
```
(179, 1)
```

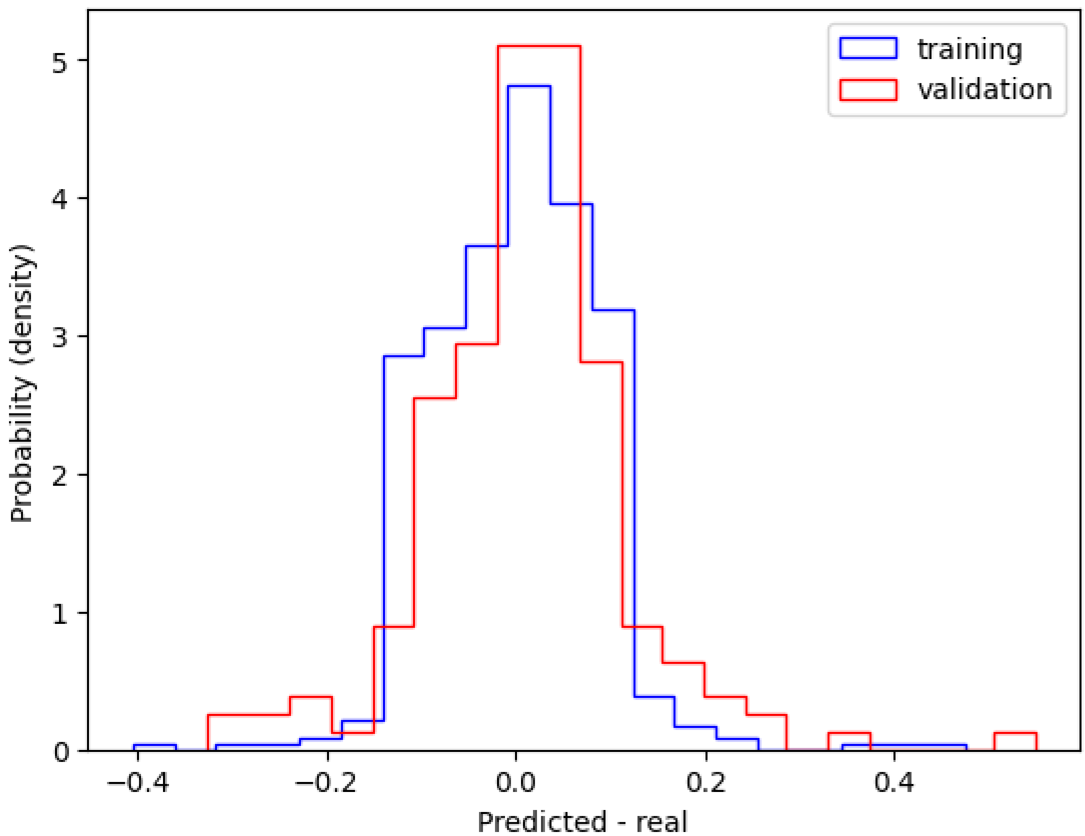
In [20]:
```
# Inspect the distribution of the difference between the predicted and actual

plt.hist(Y_tr_pred-Y_tr,20,color='b',histtype='step',label='training')
plt.hist(Y_va_pred-Y_va,20,color='r',histtype='step',label='test')
plt.xlabel('Predicted - real')
plt.ylabel('N')
plt.legend()
```

Out[20]: <matplotlib.legend.Legend at 0x24e1012ffa0>



In [21]:
```
# If the size validation set and training set are different,
# it would be better to normalize the distributions for a better comparison.

plt.hist(Y_tr_pred-Y_tr,20,color='b',histtype='step',density=True,label='trai
plt.hist(Y_va_pred-Y_va,20,color='r',histtype='step',density=True,label='vali
plt.xlabel('Predicted - real')
plt.ylabel('Probability (density)')
plt.legend()
```

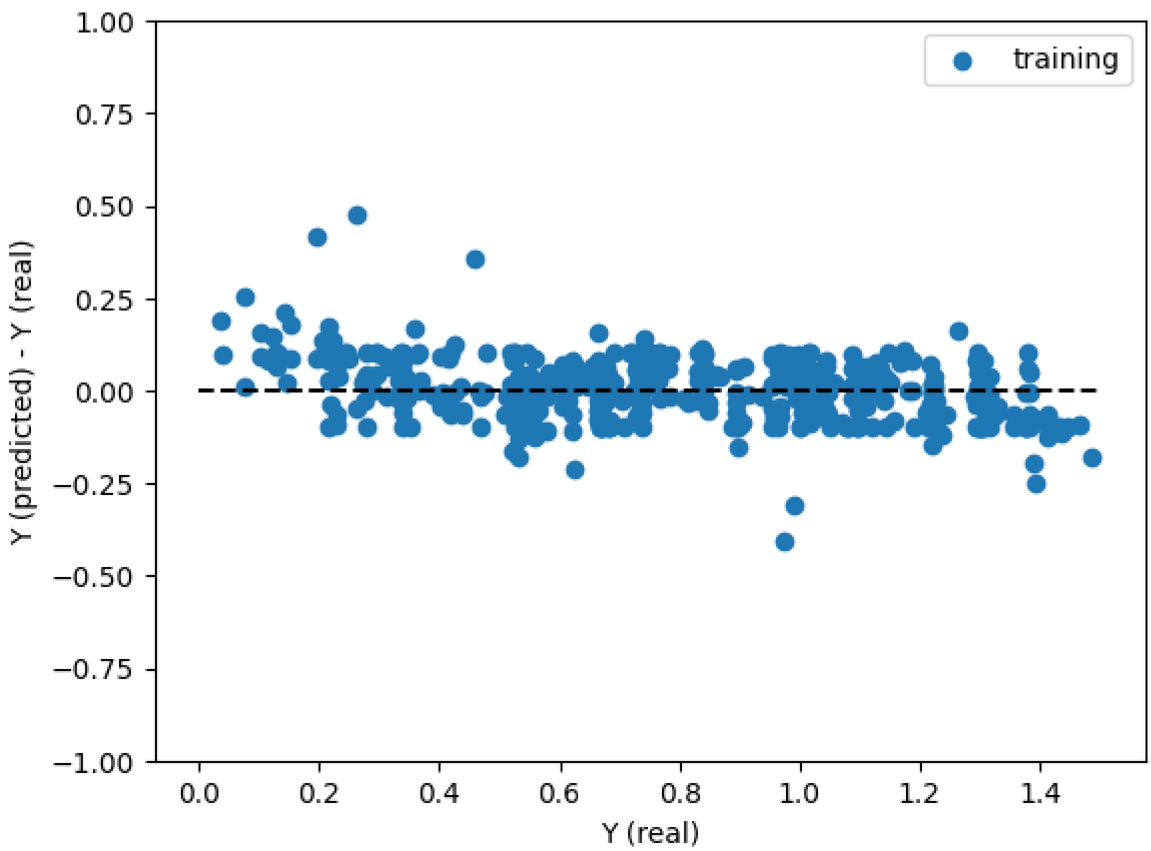
Out[21]: <matplotlib.legend.Legend at 0x24e102c3a30>



Inspecting systematic errors. The best is to have asymmetric Gaussian density around the dashed line

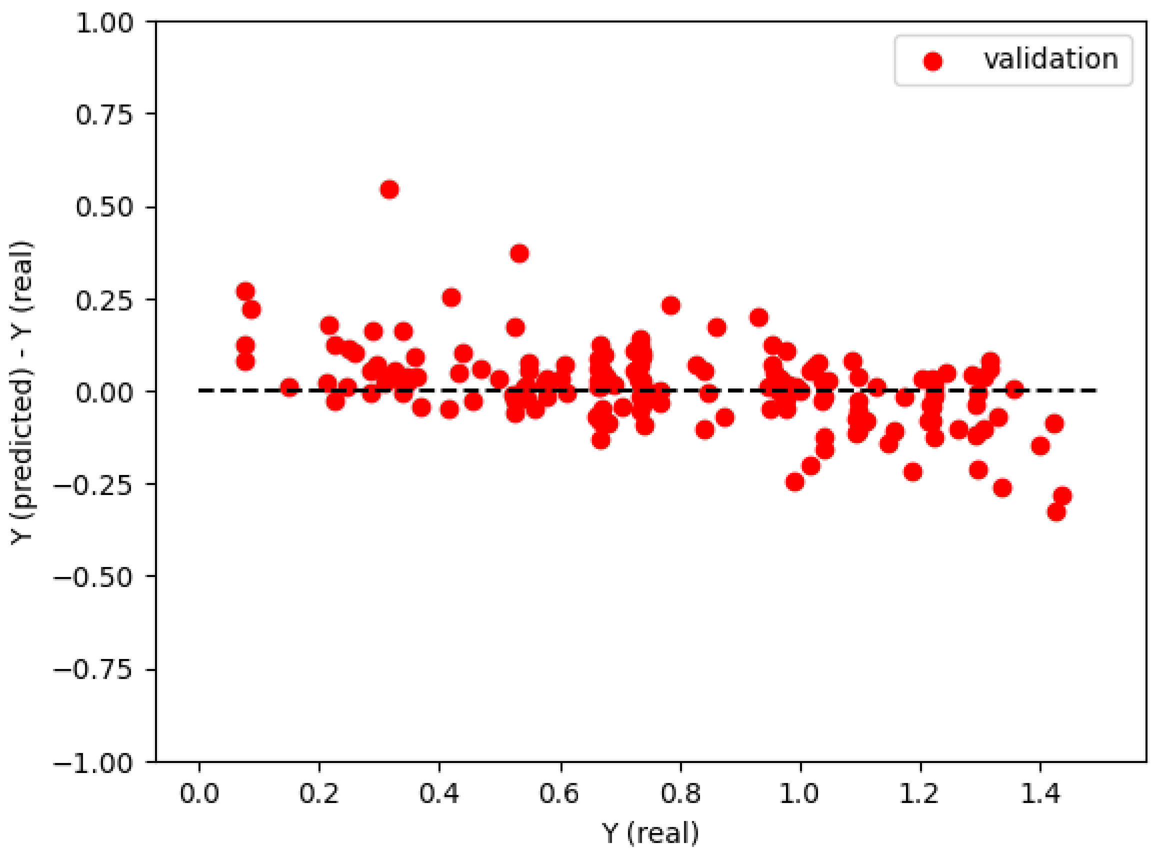
In [22]:
```
# Inspecting systematic errors for the training set
plt.scatter(Y_tr,Y_tr_pred-Y_tr,label='training')
plt.xlabel('Y (real)')
plt.ylabel('Y (predicted) - Y (real)')
plt.plot([0,1.5],[0,0],'--k')
plt.ylim([-1,1])
plt.legend()
```

Out[22]:
```
<matplotlib.legend.Legend at 0x24e102d8fa0>
```



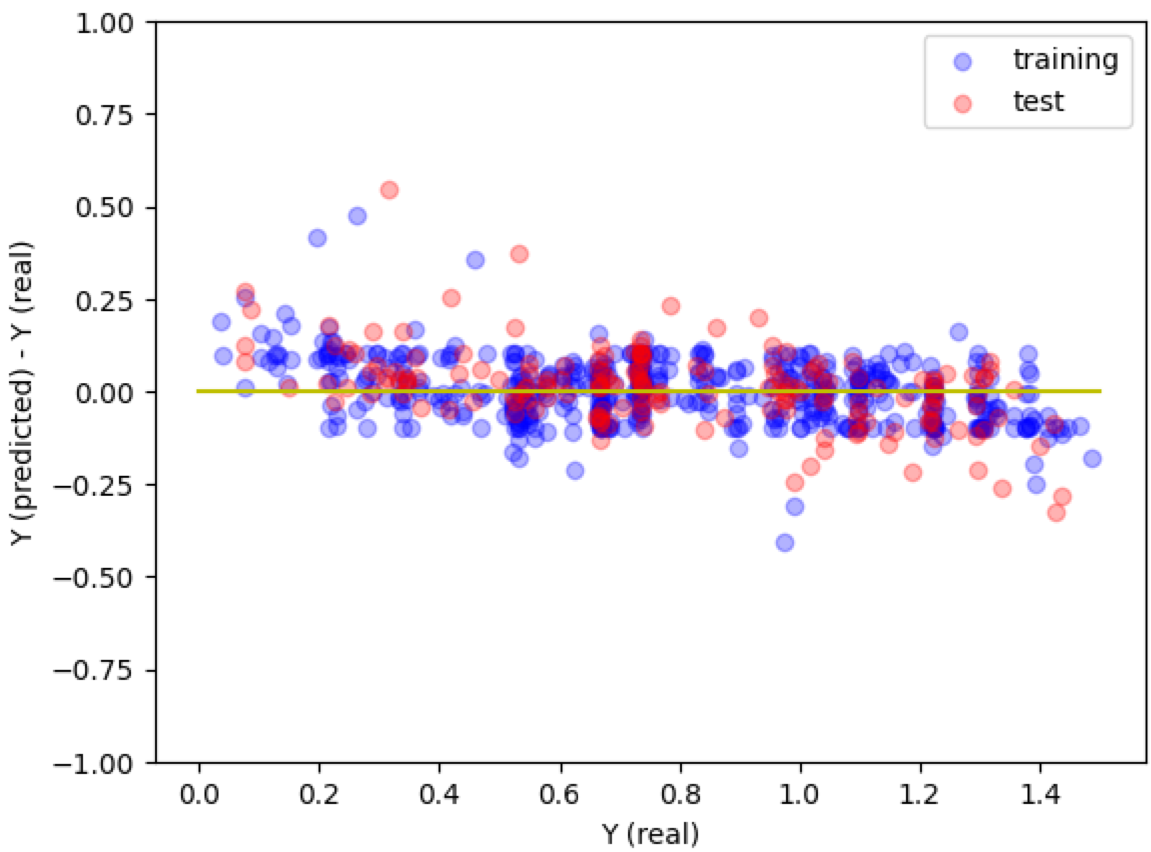
In [23]:
```
# Inspecting systematic errors for the validation set
plt.scatter(Y_va,Y_va_pred-Y_va,label='validation',color='r')
plt.xlabel('Y (real)')
plt.ylabel('Y (predicted) - Y (real)')
plt.plot([0,1.5],[0,0],'--k')
plt.ylim([-1,1])
plt.legend()
```

Out[23]: <matplotlib.legend.Legend at 0x24e102d71f0>



In [24]:
```
# Inspecting systematic errors for the training set and validation set togeth

plt.scatter(Y_tr,Y_tr_pred-Y_tr,label='training',color='b',alpha=.3)
plt.scatter(Y_va,Y_va_pred-Y_va,label='test',color='r',alpha=.3)
plt.xlabel('Y (real)')
plt.ylabel('Y (predicted) - Y (real)')
plt.plot([0,1.5],[0,0],'y')
plt.ylim([-1,1])
plt.legend()
```

Out[24]:
```
<matplotlib.legend.Legend at 0x24e0ebcae20>
```



In [ ]:

In [ ]: