2.2 Kaggle Housing Competition

November 3, 2021

1 Participate in the Kaggle Housing Prices Advanced Regression competition

We have already prepared the the dataset, but for more info on the competition, go to: https://www.kaggle.com/c/house-prices-advanced-regression-techniques/overview

The below code will only show you how to load the training set, you will have to add code to learn a model. Cross validate the training set to check for under- and overfitting.

Then finally, there is some code to generate a csv file that you can submit to the Kaggle leaderboard.

[1]:
```
from pipetorch.data import housing_prices_kaggle
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from numpy import sqrt, std, mean, arange
```

[2]:
```
df = housing_prices_kaggle()
```

2 Test set

In this case, the PipeTorch DataFrame contains a train and test set. The train set contains 'labels' (Data Science slang for target variable), while the test set does not. In PipeTorch, you will see the entire set though when you inspect the DataFrame, but for the test set the target variable is set to NaN. You can address these subsets as `df.train` and `df.test` which will just give you resp. the train and test parts.

[3]:
```
df
```

[3]:
```
        Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
0        1          60       RL         65.0     8450   Pave   NaN      Reg
1        2          20       RL         80.0     9600   Pave   NaN      Reg
2        3          60       RL         68.0    11250   Pave   NaN      IR1
3        4          70       RL         60.0     9550   Pave   NaN      IR1
4        5          60       RL         84.0    14260   Pave   NaN      IR1
```

```
…     …            …       …            …       …    …    …       …
2914  2915        160       RM         21.0    1936   Pave   NaN       Reg
2915  2916        160       RM         21.0    1894   Pave   NaN       Reg
2916  2917         20       RL        160.0   20000   Pave   NaN       Reg
2917  2918         85       RL         62.0   10441   Pave   NaN       Reg
2918  2919         60       RL         74.0    9627   Pave   NaN       Reg

     LandContour Utilities  … PoolArea PoolQC  Fence MiscFeature MiscVal  \
0            Lvl    AllPub  …        0    NaN    NaN         NaN       0
1            Lvl    AllPub  …        0    NaN    NaN         NaN       0
2            Lvl    AllPub  …        0    NaN    NaN         NaN       0
3            Lvl    AllPub  …        0    NaN    NaN         NaN       0
4            Lvl    AllPub  …        0    NaN    NaN         NaN       0
…           …        …  …      …     …      …          …      …
2914         Lvl    AllPub  …        0    NaN    NaN         NaN       0
2915         Lvl    AllPub  …        0    NaN    NaN         NaN       0
2916         Lvl    AllPub  …        0    NaN    NaN         NaN       0
2917         Lvl    AllPub  …        0    NaN  MnPrv        Shed     700
2918         Lvl    AllPub  …        0    NaN    NaN         NaN       0

     MoSold YrSold  SaleType  SaleCondition  SalePrice
0         2   2008        WD         Normal   208500.0
1         5   2007        WD         Normal   181500.0
2         9   2008        WD         Normal   223500.0
3         2   2006        WD        Abnorml   140000.0
4        12   2008        WD         Normal   250000.0
…       …     …        …            …          …
2914      6   2006        WD         Normal        NaN
2915      4   2006        WD        Abnorml        NaN
2916      9   2006        WD        Abnorml        NaN
2917      7   2006        WD         Normal        NaN
2918     11   2006        WD         Normal        NaN

[2919 rows x 81 columns]
```

3 Cross Validation

To validate the model and tune hyperparameters, you will still need a validation set. When you `split(valid_perc)` the DataFrame, you will only split the training set into a train and valid part.

Having the train, valid and test set in a single DataFrame has a few advantages: - any preprocessing you do will be the same on all subsets (as it probably should). - when you scale the data, you should learn a scaler on the training set only, then apply the same scaler on the validation and test set as well. PipeTorch automatically takes care of this.

4 Prepare the data

I this case, you may not want to slice out the columns you do not use. When we submit the results to Kaggle, we need the `Id`'s of these houses, but we do not want to use that as an input feature. You can use `columnx()` and `columny()` to keep all columns, but use only those features in your experiment. You can also invert the feature selection by using `columnx(omit=True)` and the target variables are omitted by default.

[4]:
```
data = df.columnx('LotArea').columny('SalePrice').split(0.3, random_state=0).
↪scalex()
```

Add your code to fit a Linear Regression model. Start with just LotArea as input and SalePrice as target variable. If you want, you can add another numerical feature later, but do pick one that make sense (for example, Id would probably not make sense, why?).

Note: `data.train_X` is the same as `data.train.X`. In fact, `data.train` will give you the training subset of the PipeTorch DataFrame that is ready to generate the data for use with SKLearn (e.g. create the X and y, but also make plots). When you call `X` or `y`, the specified preprocessing is performed (selecting the features, scaling, categorizing, etc.).

5 Linear Regression single variable

[5]:
```
model = LinearRegression()
model.fit(data.train_X, data.train_y)
```

[5]:
```
LinearRegression()
```

[6]:
```
sqrt(mean_squared_error(data.valid_y, model.predict(data.valid_X)))
```

[6]:
```
73737.07338057815
```

[7]:
```
std(data.valid_y)
```

[7]:
```
74288.77400280123
```

5.1 Polynomials for single variable

See what happens when you add polynomials and draw a validation curve to see which value for polynomial degree is best.

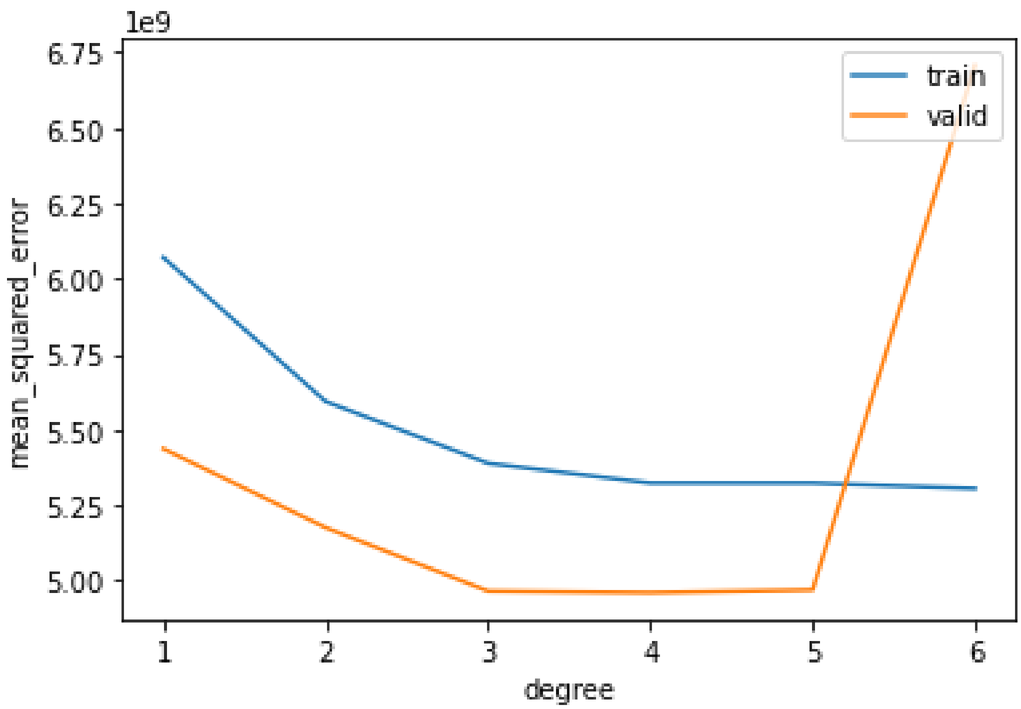
Note, when using higher-order polynomials, the model is often improved when the data is **scaled**. However, when the target variable is also scaled this no longer reflects salesprices on a similar scale. The most simple solution for now is to not scale the target variable by using `scalex()` instead of `scale()`. Alternatively, we could inverse the scaling transformation, but we will see how to do that later in this course.

```
[8]: data = df.columnx('LotArea').columny('SalePrice').split(0.3, random_state=0).
     ↪scalex()
```

```
[9]: evaluate = data.evaluate(mean_squared_error)

     for degree in range(1,7):
         model = LinearRegression()
         evaluate.run_sklearn(model,degree=degree,df=data.polynomials(degree))
```

```
[10]: evaluate.line_metric('degree')
```




```
[11]: p = data.polynomials(degree=5)
      model.fit(p.train_X,p.train_y)
      model.score(p.valid_X,p.valid_y)
```

[11]: 0.09951809073503215

```
[12]: sqrt(mean_squared_error(p.valid_y, model.predict(p.valid_X)))
```

[12]: 70495.38509704951

5.2 Ridge for single variable

Replace the Linear Regression model with a Ridge (or Lasso) model. Does that improve the results?

These regularized versions of Linear Regression have a hyperparameter to control the amount of regularization. Make sure to tune that.

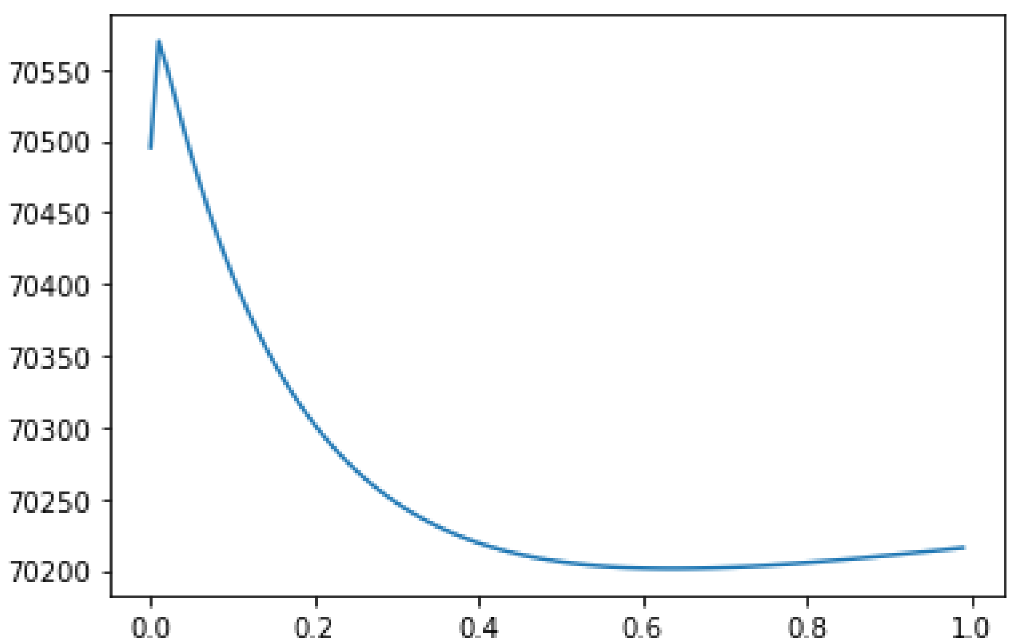
Load the training set. SalePrice is the target variable. You do not have to use all variables, a suggestion is to start with LotArea as feature.

[13]:
```
alphas = arange(0, 1, 0.01)
A = []
S = []

for alpha in alphas:
    model = Ridge(alpha=alpha)
    model.fit(p.train_X,p.train_y)
    A.append(alpha)
    S.append(sqrt(mean_squared_error(p.valid_y, model.predict(p.valid_X))))
```

[14]:
```
dfRidge = pd.DataFrame(data={'Alpha':A,'MRS':S})
plt.plot(dfRidge['Alpha'],dfRidge['MRS'])
print(dfRidge[dfRidge.MRS == dfRidge.MRS.min()])
```

```
    Alpha           MRS
64   0.64  70201.306414
```



[15]:
```
model = Ridge(alpha=0.64)
model.fit(p.train_X,p.train_y)
model.score(p.train_X,p.train_y)
```

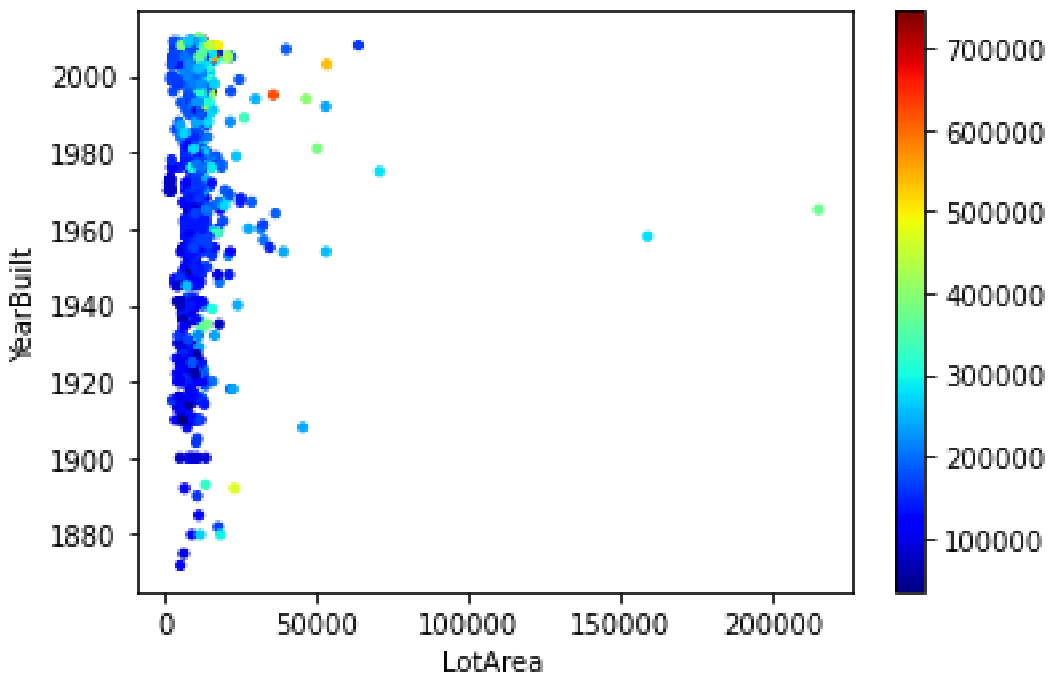
[15]: 0.18984582988771126

[16]:
```
sqrt(mean_squared_error(p.valid_y, model.predict(p.valid_X)))
```

[16]: 70201.30641360089

6 Multivariate Regression

[17]:
```
data = df.columnx('LotArea','YearBuilt').columny('SalePrice').split(0.3,␣
↪random_state=0).scalex()
data.train.scatter2d_color(s=10)
```



[18]:
```
model = LinearRegression()
model.fit(data.train_X,data.train_y)
```

[18]: LinearRegression()

[19]:
```
model.score(data.valid_X,data.valid_y)
```

[19]: 0.2951883468604991

[20]:
```
sqrt(mean_squared_error(data.valid_y, model.predict(data.valid_X)))
```

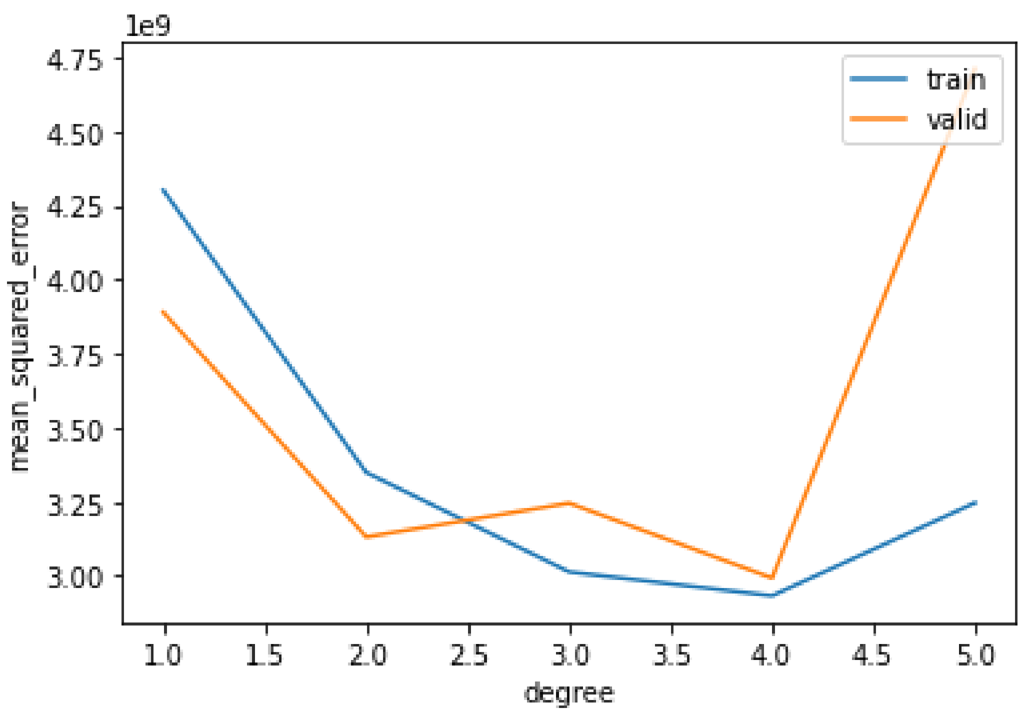
[20]: 62367.70010923204

6.1 Polynomials for multivariable

[21]:
```
evaluate = data.evaluate(mean_squared_error)

for degree in range(1,6):
    model = LinearRegression()
    evaluate.run_sklearn(model,degree=degree,df=data.polynomials(degree))
```

[22]:
```
evaluate.line_metric('degree')
```



[23]:
```
p = data.polynomials(degree=4)
model.fit(p.train_X,p.train_y)
model.score(p.valid_X,p.valid_y)
```

[23]: 0.4577430635474212

[24]:
```
sqrt(mean_squared_error(p.valid_y, model.predict(p.valid_X)))
```

[24]: 54704.83963555048

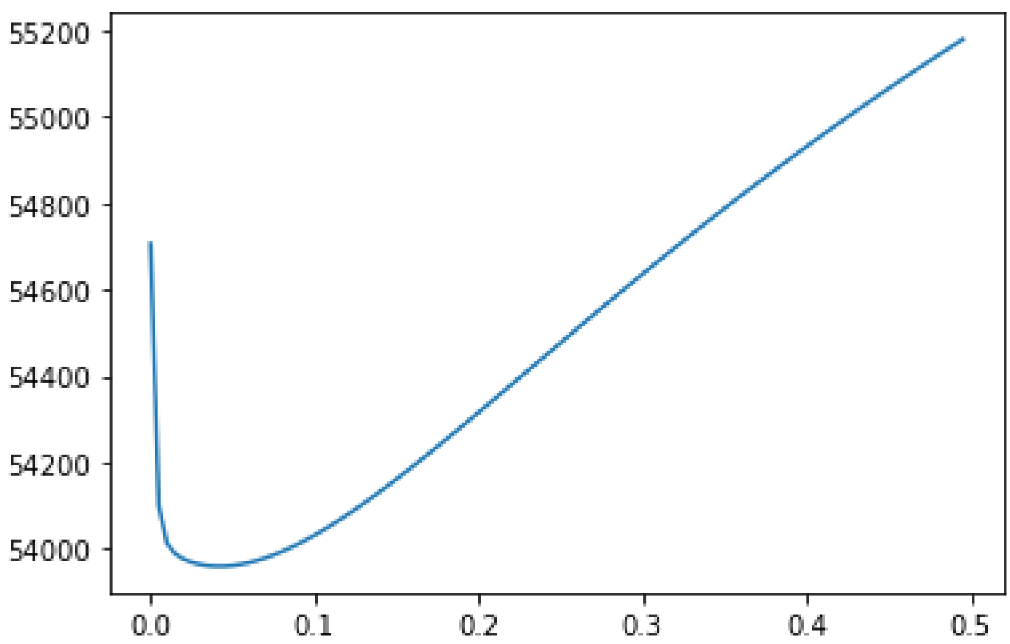
6.2 Ridge for multivariable and polynomials

[25]:
```
alphas = arange(0, 0.5, 0.005)
A = []
S = []

for alpha in alphas:
    model = Ridge(alpha=alpha)
    model.fit(p.train_X,p.train_y)
    A.append(alpha)
    S.append(sqrt(mean_squared_error(p.valid_y, model.predict(p.valid_X))))
```

[26]:
```
dfRidge = pd.DataFrame(data={'Alpha':A,'MRS':S})
plt.plot(dfRidge['Alpha'],dfRidge['MRS'])
print(dfRidge[dfRidge.MRS == dfRidge.MRS.min()])
```

```
   Alpha           MRS
8   0.04  53960.413453
```



[27]:
```
model = Ridge(alpha=0.04)
model.fit(p.train_X,p.train_y)
model.score(p.train_X,p.train_y)
```

[27]: 0.5298477093858949

[28]:
```
sqrt(mean_squared_error(p.valid_y, model.predict(p.valid_X)))
```

[28]: 53960.413453271656

7 Submission

To submit you results, you have to create a .csv file for the test set. You can follow these steps. You can generate predictions on the test set the same way you do for the train and valid set. You can fill in these results in the `SalePrice` column of the test set and use the `.to_csv` to export what is needed to a .csv file.

[29]:
```
# generate predictions of the model that was trained on the training set
# do not learn a new model here!
y_pred = model.predict(p.test_X)
```

[30]:
```
# add the predictions as a new column SalePrice
results = data.test.add_column(y_pred, 'SalePrice')
```

[31]:
```
# the competition requires that you create a .csv file to submit
# have not tested this, but this should work
results[['Id', 'SalePrice']].to_csv('boston_housing_results.csv', index=False,␣
↪header=True)
```

After running the last cell, you can find and download your .csv file in Jupyter's file list. Submit it to see what your score is on the leaderboard!!!