Branch: **master** | **kaggle-competitions** / **README.md** | **Find file** | **Copy path**

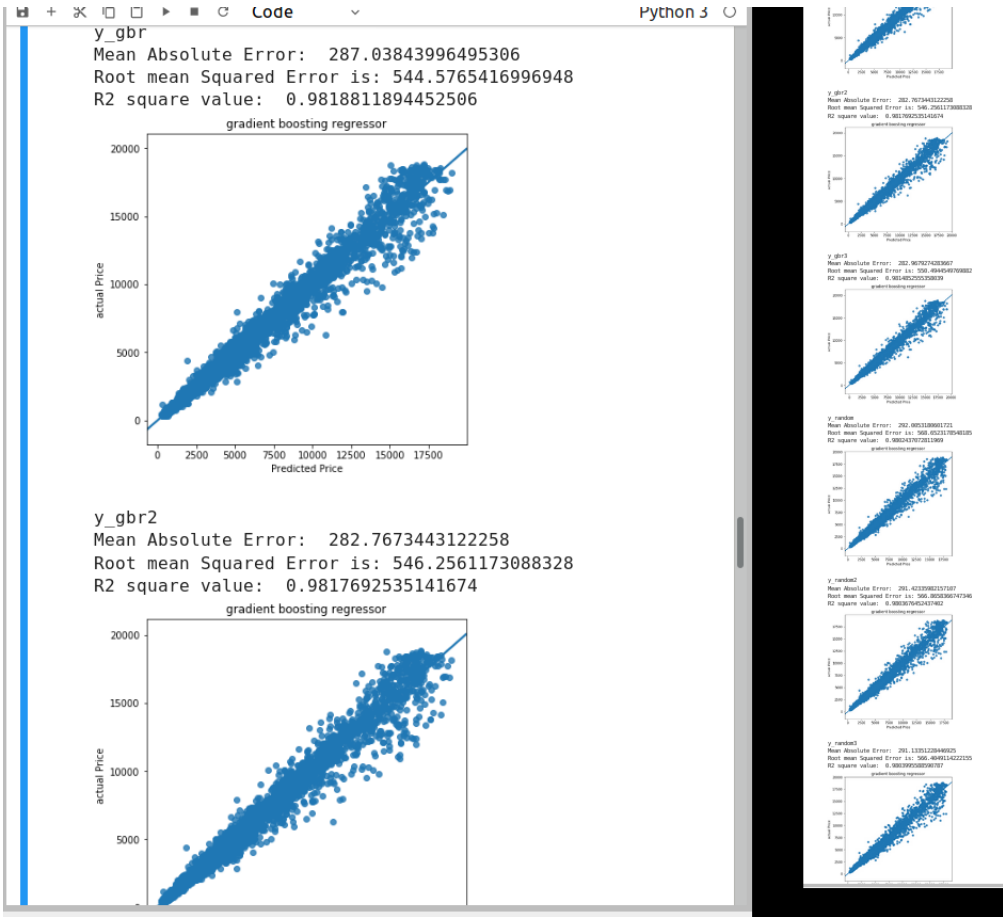
albertovpd Update README.md 7b4e204 15 minutes ago

1 contributor

51 lines (34 sloc) 2.92 KB | **Raw** | **Blame** | **History**

1st kaggle-competition

Or "How I spent 2 days learning a lot and getting frustrating low ranks"



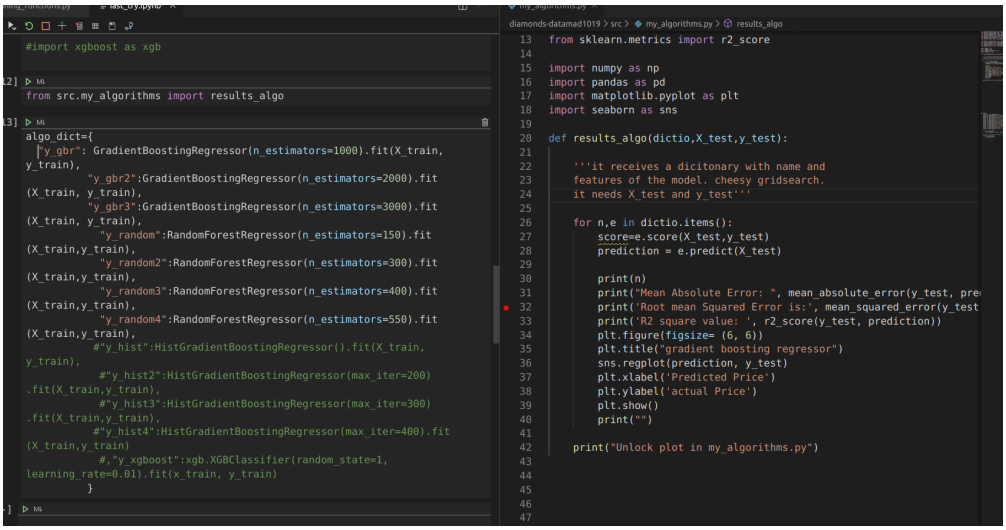
The proccess

- I tried to apply some logic and don't try autoML or similar tools

- Cleaning job

- Find documentation about the problem
- Look in every column the type and elemnts

- Correlation matrix to find the proper features
- Turn categorical columns into numeric ones
  - At first I tried with numpy to give slight different weights to elements with numpy. In the end I just gave int types. -py to clean both datasets



```
#import xgboost as xgb

from src.my_algorithms import results_algo

algo_dict={
    "y_gbr": GradientBoostingRegressor(n_estimators=1000).fit(X_train,
y_train),
           "y_gbr2":GradientBoostingRegressor(n_estimators=2000).fit
(X_train, y_train),
           "y_gbr3":GradientBoostingRegressor(n_estimators=3000).fit
(X_train, y_train),
           "y_random":RandomForestRegressor(n_estimators=150).fit
(X_train,y_train),
           "y_random2":RandomForestRegressor(n_estimators=300).fit
(X_train,y_train),
           "y_random3":RandomForestRegressor(n_estimators=400).fit
(X_train,y_train),
           "y_random4":RandomForestRegressor(n_estimators=550).fit
(X_train,y_train),
           #"y_hist":HistGradientBoostingRegressor().fit(X_train,
y_train),
           #"y_hist2":HistGradientBoostingRegressor(max_iter=200)
.fit(X_train,y_train),
           #"y_hist3":HistGradientBoostingRegressor(max_iter=300)
.fit(X_train,y_train),
           #"y_hist4":HistGradientBoostingRegressor(max_iter=400).fit
(X_train,y_train)
           #,"y_xgboost":xgb.XGBClassifier(random_state=1,
learning_rate=0.01).fit(x_train, y_train)
           }
```

```
diamonds-datamad1019 > src > my_algorithms.py > results_algo
from sklearn.metrics import r2_score

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

def results_algo(dictio,X_test,y_test):

    '''it receives a dicitonary with name and
    features of the model. cheesy gridsearch.
    it needs X_test and y_test'''

    for n,e in dictio.items():
        score=e.score(X_test,y_test)
        prediction = e.predict(X_test)

        print(n)
        print("Mean Absolute Error: ", mean_absolute_error(y_test, pre
        print('Root mean Squared Error is:', mean_squared_error(y_test
        print('R2 square value: ', r2_score(y_test, prediction))
        plt.figure(figsize= (6, 6))
        plt.title("gradient boosting regressor")
        sns.regplot(prediction, y_test)
        plt.xlabel('Predicted Price')
        plt.ylabel('actual Price')
        plt.show()
        print("")

    print("Unlock plot in my_algorithms.py")
```

- https://github.com/albertovpd/kaggle-competitions/blob/master/diamonds-datamad1019/output/pics/printing%20function.png

- Remove outliers from the dataset to have a more accurate model

```
onds-datamad1019 > src > my_maths.py > remove_outlier_col
import numpy as np
def remove_outlier_col(df_in, col_name):
    q1 = df_in[col_name].quantile(0.25)
    q3 = df_in[col_name].quantile(0.75)
    iqr = q3-q1 #Interquartile range
    fence_low  = q1-1.5*iqr
    fence_high = q3+1.5*iqr
    df_out = df_in.loc[(df_in[col_name] > fence_low) & (df_in[col_name] < fence_high)]
    return df_out

def remove_outliers(df):
    for e in list(df.columns):
        df=remove_outlier_col(df,"{}".format(e))
    return df

def standarizing(df):
    for col in df.columns:
        df[col] = (df[col] - np.mean(df[col])) / np.std(df[col])
    return df
```

- The price and weight of diamonds are exponentially related, so I give an exponential weight to the "carat" row, and removed columns strongly correlated with this one
- Standarize all my X columns with sklearn
- .py file to check 12 different configurations of algorithms, searching for the min Mean Absolute Error, Root mean Squared Error and R2 square value

alt

- .py file to clean automatically my test dataset exactly like the training dataset (yeah, I could have written a better code, deadline matters)

amonds-datamad1019 > src > cleaning_functions.py > ...

```python
def cleaning_diamonds2(df):
    # i already checked the nature of every row and its distribution



    df.cut=df.cut.replace(to_replace={"Fair":1,"Good":2,"Very Good":3,"Ideal":4,"Premium":5})
    color=list(df.color.unique())
    color.sort(reverse=True)

    letters=['D', 'E', 'F','G', 'H' , 'I', 'J']
    weight=list(range(1,8))
    weight.sort(reverse=True)
    color_dict=dict(zip(letters,weight))
    for e in df.color:
        for x,y in color_dict.items():
            if e==x:
                df.color.replace(e,y,inplace=True)

    nomenclature=["I1","SI2","SI1","VS2","VS1","VVS2","VVS1","IF"]
    weight=list(range(1,9))
    weight.sort(reverse=True)
    clarity_dict=dict(zip(nomenclature,weight))

    for e in df.clarity:
        for x,y in clarity_dict.items():
            if e==x:
                df.clarity.replace(e,y,inplace=True)

    df['volume'] = df['x'] * df['y'] * df['z']
    df.drop(["x","y","z"], axis=1,inplace=True)

    # sorting rows randomly
    #df.carat=(np.log(df.carat)).abs()
    #df.carat=df.carat*math.exp(2)

    #standarizing columns
    #columns=['carat', 'cut', 'color', 'clarity', 'depth', 'table']
    #for e in columns:
```

- Train the model with my best configuration (the first picture)

https://raw.githubusercontent.com/albertovpd/kaggle-competitions/master/diamonds-datamad1019/output/pics/final%20-results.png

- Get the predicted "y"
- Submit results

Conclusions

I really wanted to try my skills as scientist in my first "fight" with ML. I miserably lost and learnt about it. - 1. The most important task is to study the field and clean your dataset properly. - 2. Do not give weight to columns without knowlegde. If you are going to do random stuff, you have way better tools for that purpose. - 3. Do not use Scikit GridSearchCV randomly. - 4. Avoid the bad use of H2O's autoML and tools like that. It Depends on the size and type of dataset. Nevertheless, if you don't have the proper time to study the field your working with, go back to 2 and do it. Also very useful when you think you can't optimize your result anymore.

```
Those are my principles, and if you don't like them... Well, I have
others.
```