Comparison of Regressors

Prof.Dr. Bahadır AKTUĞ
Machine Learning with Python

**Compiled from sources given in the references.*

Regressor Algorithms

- A simple comparison will be made for regression algorithms.
- The algorithms to be used:
  - Ridge Regression
  - KNN (K Nearest Neighbors)
  - Bayesian Regression
  - Decision Tree Regression
  - SVM (Support Vector Machine) Regression
- It should be noted that this is just a demonstration and highly data-dependent, i.e. different performance output would be obtained with different data.
- It is only through trial and error and checking the performance metrics, we can narrow down and pick certain algorithms.

Data (IMDB Reviews)

```
color
director_name
num_critic_for_reviews
duration
director_facebook_likes
actor_3_facebook_likes
actor_2_name
actor_1_facebook_likes
gross
genres
actor_1_name
movie_title
num_voted_users
cast_total_facebook_likes
actor_3_name
facenumber_in_poster
plot_keywords
movie_imdb_link
num_user_for_reviews
language
country
content_rating
budget
title_year
actor_2_facebook_likes
imdb_score
aspect_ratio
movie_facebook_likes
```

- International Movie Databse (IMDB) ratings and metadata
- 5043 records
- 28 parameters as given in the left
- For this study, we'll assume a dependent parameter: imdb_score
- All other parameters will be assumed independent
- In this regression exercise, we'll try to estimate the imdb_score based on other parameters

Step-1: Load Data

- The data can be loaded as a Pandas Dataframe.
- Some pre-processing is needed since
  - there are null values to be filled
  - data should be scaled (standardized)
  - some features are non-numerical, non-categorical
  - dimension reduction needs to be applied

```
import pandas as pd
from pandas import DataFrame,Series
f = pd.read_csv("movie_metadata.csv")
data=DataFrame(f)
```

Step 2: Preprocessing

- Some columns are strings. Strings are imported as datatype «object» that we cannot use directly in regression, so they need to be eliminated.

```
X_data=data.dtypes[data.dtypes!='object'].index
X_train=data[X_data]
```

```
>>> data.dtypes[data.dtypes=='object']
color                   object
director_name           object
actor_2_name            object
genres                  object
actor_1_name            object
movie_title             object
actor_3_name            object
plot_keywords           object
movie_imdb_link         object
language                object
country                 object
content_rating          object
dtype: object
```

```
>>> data.dtypes[data.dtypes!='object']
num_critic_for_reviews          float64
duration                        float64
actor_3_facebook_likes          float64
actor_1_facebook_likes          float64
gross                           float64
num_voted_users                   int64
cast_total_facebook_likes         int64
facenumber_in_poster            float64
num_user_for_reviews            float64
budget                          float64
title_year                      float64
actor_2_facebook_likes          float64
imdb_score                      float64
aspect_ratio                    float64
movie_facebook_likes              int64
dtype: object
```

Step 2: Preprocessing (continued)

- Now, we need to handle null values

```
X_train=X_train.fillna(0)
```

- The dependent variable (y) will be assigned the imdb score:

```
y=X_train['imdb_score']
```

- Finally, we need to drop imdb score column from our independent variable set (X):

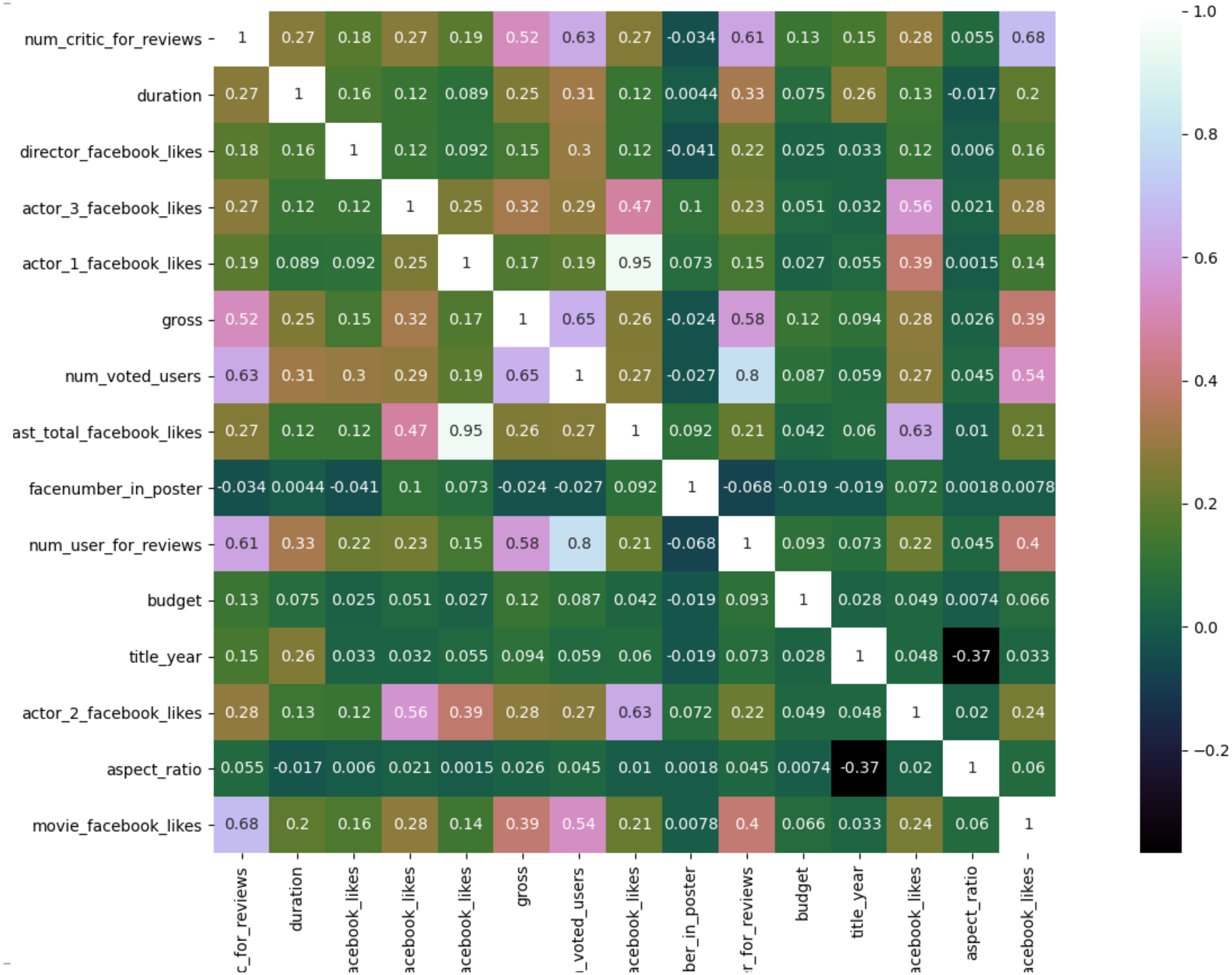
```
X_train.drop(['imdb_score'],axis=1,inplace=True)
```

Step 3: Checking the correlation of Features

- When we have too many attributes, it is useful to check the correlation of features
- In this way, redundant features can be excluded from the analysis
- This step is also necessary to check for possible linear dependency of the features which will weaken the solution.

```
corr_mat=X_train.corr(method='pearson’)
plt.figure(figsize=(20,10))
sns.heatmap(corr_mat,vmax=1,square=True,annot=True,cmap='cubehelix')
```

Step 3: Checking the correlation of Features



Step 4: Scaling Features

- Due to possible scale difference between features, a scaling is necessary.
- The easiest way to do it to use StandardScaler function of sci-kit learn module. StandardScaler.
- StandardScaler standardizes a feature by subtracting the mean and then scaling to unit variance. Unit variance means dividing all the values by the standard deviation.

```
from sklearn.preprocessing import StandardScaler
X_Train=X_train.values
X_Train=np.asarray(X_Train)
X_std=StandardScaler().fit_transform(X_Train)
```

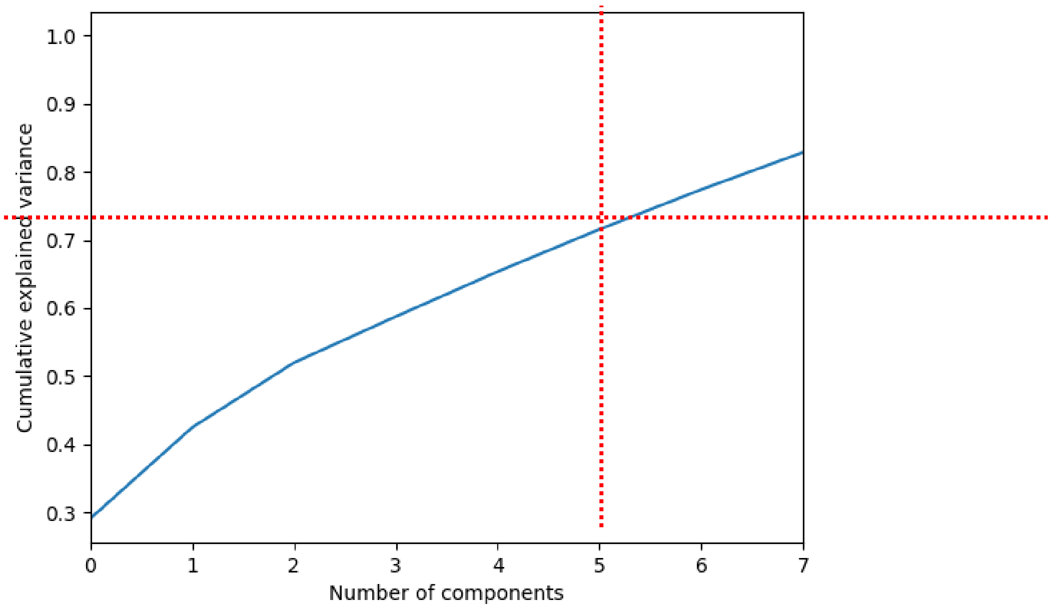
Step 5: Dimension Reduction

- It is useful to reduce the dimension since it improves the solution and help getting rid of noise.
- For this purpose, PCA (Principal Component Analysis) will be employed.
- Features which will maximize the variation the most will be used instead of the whole set.

```
from sklearn.decomposition import PCA
pca = PCA().fit(X_std)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlim(0,7,1)
plt.xlabel('Number of components')
plt.ylabel('Cumulative explained variance')
```

Step 5: Dimension Reduction

- First five components explain the data more than 70%
- Addition of component contribute only marginally. Thus, only first 5 components will be used.




```
from sklearn.decomposition import PCA
sklearn_pca=PCA(n_components=5)
X_Train=sklearn_pca.fit_transform(X_std)
```

Step 6: Split Data for training and test

- One fundamental step is the splitting data (target as well) as training and test.
- In this way, the model is trained with the training data then tested with the test data.
- Since test data is not used during training, it is considered as a more reliable assessment of the model.
- Scikit-learn provides a utility for this splitting:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X_train, y,
test_size=0.25, random_state=0)
```

Step 5: Parameters & Results

- The input parameters of the models are as follows:

  Ridge Regression
  KNN (K Nearest Neighbors)
  Bayesian Regression
  Decision Tree Regression
  SVM (Support Vector Machine) Regression

- The performance results are as follows:

```
Logistic Regression: Mean Accuracy = 82.75% - SD Accuracy = 11.37%
K Nearest Neighbor: Mean Accuracy = 90.50% - SD Accuracy = 7.73%
Kernel SVM: Mean Accuracy = 90.75% - SD Accuracy = 9.15%
Naive Bayes: Mean Accuracy = 85.25% - SD Accuracy = 10.34%
Decision Tree: Mean Accuracy = 84.75% - SD Accuracy = 7.86%
Random Forest: Mean Accuracy = 88.25% - SD Accuracy = 8.44%
```

References

1 https://scikit-learn.org/
2 https://towardsdatascience.com/
3 McKinney, W. (2017). Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython 2nd Edition.
4 Albon, C. (2018). Machine Learning with Python Cookbook: Practical Solutions from Preprocessing to Deep Learning
5 Géron, A. (2017). Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems 1st Edition
6 Müller, A. C., Guido, S. (2016). Introduction to Machine Learning with Python: A Guide for Data Scientists
7 Burkov, A. (2019). The Hundred-Page Machine Learning Book.
8 Burkov, A. (2020). Machine Learning Engineering.
9 https://www.kaggle.com/ankitjha/comparing-regression-models/notebook
10 https://towardsdatascience.com
11 https://towardsdatascience.com/machine-learning-project-17-compare-classification-algorithms-87cb50e1cb60
12 https://machinelearningmastery.com/compare-machine-learning-algorithms-python-scikit-learn/
13 https://developers.google.com/edu/python/
14 http://learnpythonthehardway.org/book/