MATH 156 Final Project

Predicting videogame sales with various models

Group 5

University of California, Los Angeles

August 2, 2020

Table of Contents

Table of Contents

Preproccessing Data

- The main consideration we had when cleansing the data was to maintain its integrity and the integrity of our model. We ran through different ways to clean and sort data, as well as group data. Ultimately, we converged on the result that gave us the lowest RMSE.

	Platform	Year_of_Release	Genre	Publisher	Global_Sales	Critic_Score	Critic_Count	User_Score	User_Count	Developer	Rating
0	Wii	2006.0	Sports	Nintendo	82.53	76.0	51.0	8	322.0	Nintendo	E
1	NES	1985.0	Platform	Nintendo	40.24	NaN	NaN	NaN	NaN	NaN	NaN
2	Wii	2008.0	Racing	Nintendo	35.52	82.0	73.0	8.3	709.0	Nintendo	E
3	Wii	2009.0	Sports	Nintendo	32.77	80.0	73.0	8	192.0	Nintendo	E
4	GB	1996.0	Role-Playing	Nintendo	31.37	NaN	NaN	NaN	NaN	NaN	NaN

Figure: First 5 lines of data set after removing Sales Data apart from Global Sales

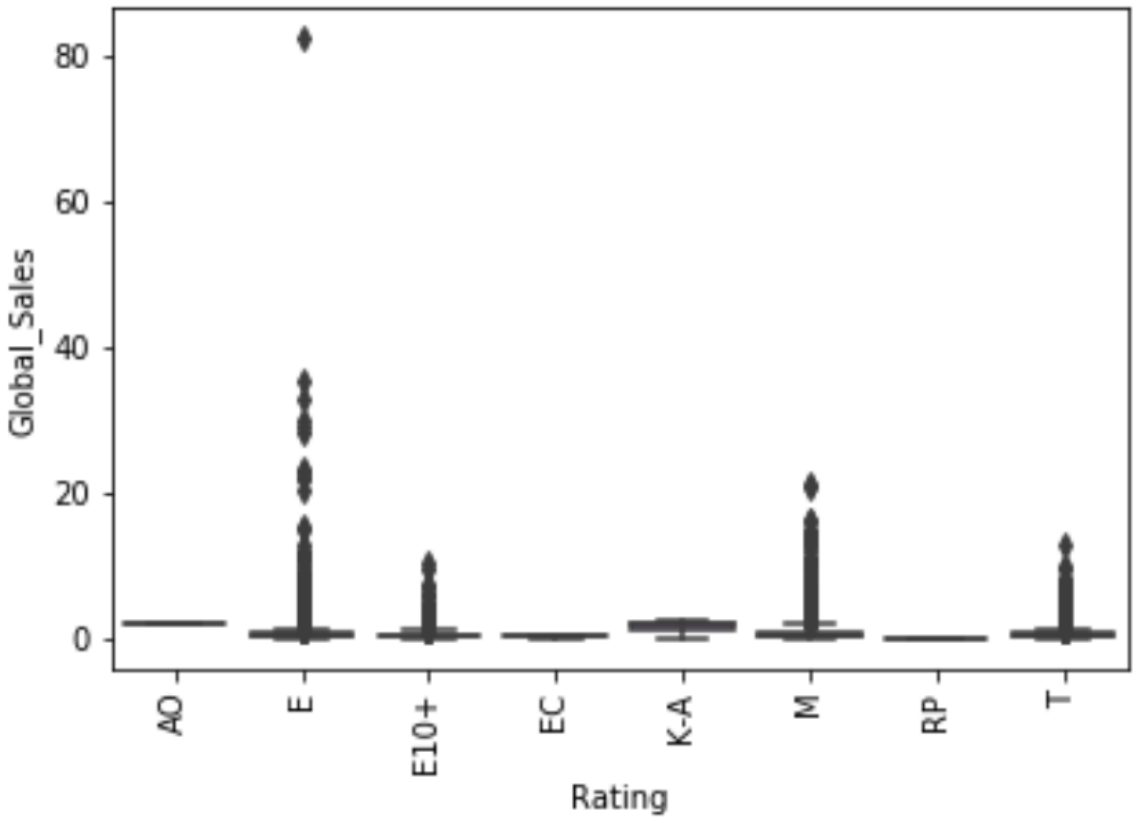
Cleansing the data

Steps:

1. Remove the rows containing missing data that cannot be imputed. (Platform, Genre, Publisher, Year of Release)
2. Replace the missing values with the median of the column. (Critic Score, User Score, Critic Count, User Count)
3. Replace categorical data with dummy variables (one-hot-encoding) (Genre, Publisher) and drop one column of each to decorrelate the columns.

Dropping categorical data

- Dropping [Rating, Publisher, Japan Sales, EU Sales, Other Sales]
- On dropping rating:



- Although there seems to exist a correlation, between rating and global sales. Our models worked better with its exclusion..so we dropped it!

Preprocessing

- In order to learn which features are important, I used a correlation matrix to plot the dependencies of variables on features like global sales.

	Year_of_Release	Global_Sales	Critic_Score	Critic_Count	User_Score	User_Count
Year_of_Release	1.000000	-0.076433	0.011411	0.223407	-0.267851	0.175339
Global_Sales	-0.076433	1.000000	0.245471	0.303571	0.088139	0.265012
Critic_Score	0.011411	0.245471	1.000000	0.425504	0.580878	0.264376
Critic_Count	0.223407	0.303571	0.425504	1.000000	0.194133	0.362334
User_Score	-0.267851	0.088139	0.580878	0.194133	1.000000	0.027044
User_Count	0.175339	0.265012	0.264376	0.362334	0.027044	1.000000

Figure: Correlation Matrix

Preprocessing

- I then plot the correlation matrix as a scatter plot to get a better idea of features to drop / include.

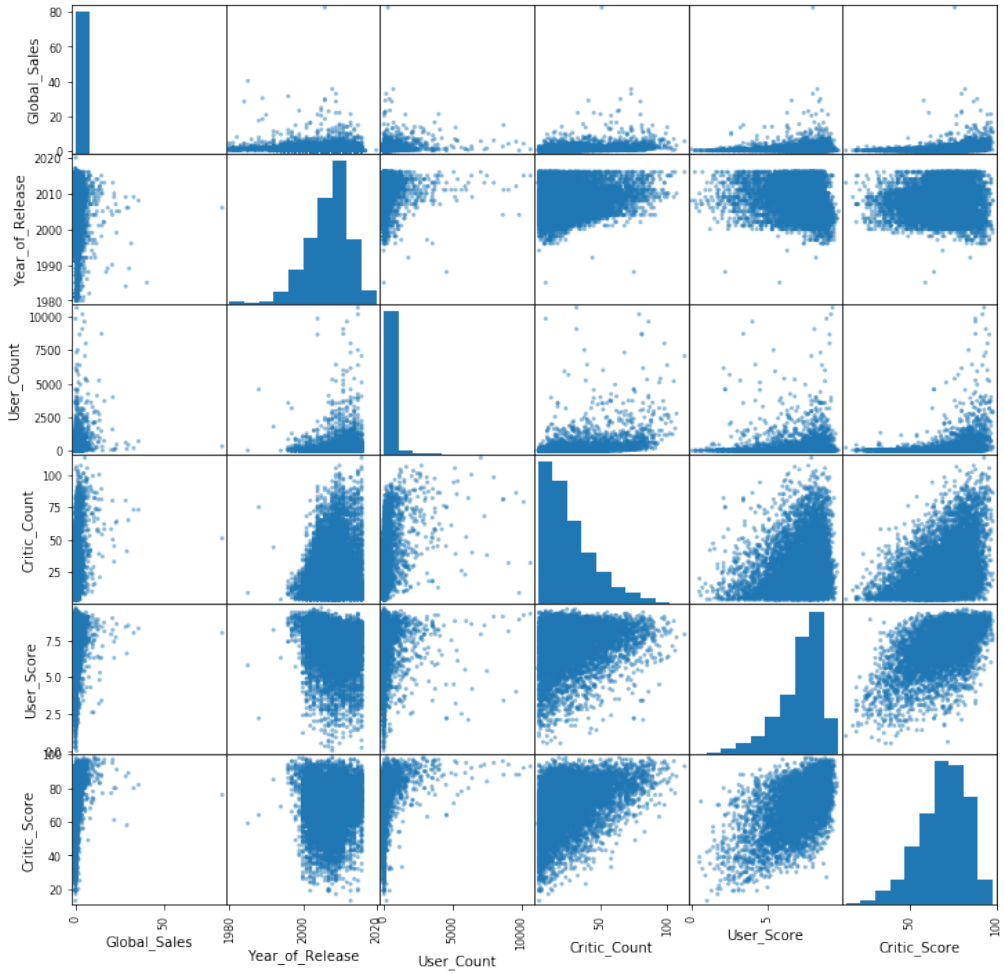


Figure: Correlation Matrix Scatter Plot

Median Imputation, Failed KNN imputation

- We replaced the NaN values with the median of the column. (Critic Score, User Score, Critic Count, User Count).

```
from sklearn.impute import SimpleImputer as Imputer
imp = Imputer(strategy="median")
attributes=["Critic_Score", "User_Score", "Critic_Count", "Critic_Count","User_Count"]
for item in attributes:
    game[item]=imp.fit_transform(game[[item]]).ravel()
```

- Attempt to KNN, not ultimately used. Using label encoder to first numerically label categorical data, then imputing using KNN.

```
features_to_label=["Platform", "Genre","Publisher", "Rating", "Developer"]

for items in features_to_label:
    game.loc[~game[items].isnull(),[items]]=labelencoder.fit_transform(game.loc[~game[items].isnull(),[
       items]])
    #Changing all features to float64
    game["Developer"]= pd.to_numeric(game["Developer"])
game["Rating"]= pd.to_numeric(game["Rating"])
#KNN
knnimp=KNNImputer()
game=knnimp.fit_transform(game)
```

Median Imputation, Failed KNN imputation

- Result of KNN Imputation was a success in regular linear regression where we saw a reduction in RMSE from 2.15 to 1.91 but did not as work well for the other models. So we did not implement this.

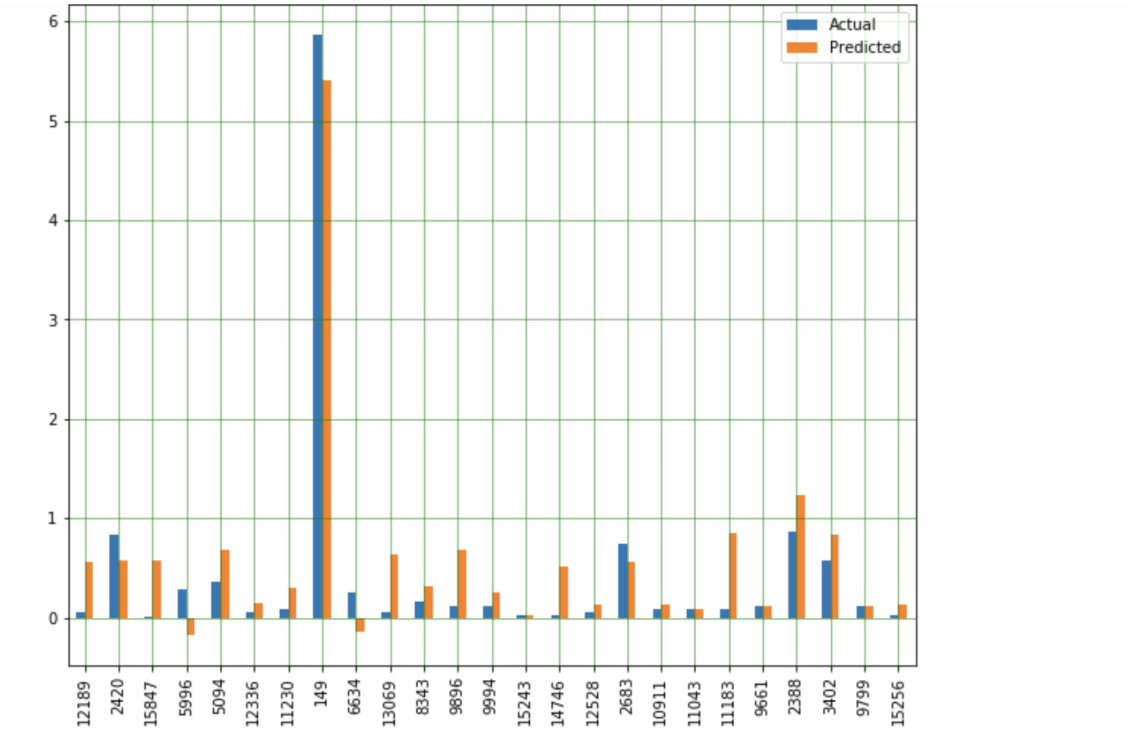


Figure: Predicted to Actual Values

Using One hot-encoding

- The why of using Onehot encoding and deletion of one column:
  Categorical data can not be used in any kind of models we wanted to implement. Lebel encoding was not sufficient. We decided to drop one column from each categorical data type because we would like to avoid non uniqueness and de-correlate the columns.
- One-hot encoding, dropping 1st column

```
    cat_features=["Rating", "Developer","Platform","Genre","Publisher"]

full_pipeline = ColumnTransformer([ # one hot encoding using this python pipeline function. very useful.
       Analogous to a design matrix
    ('cat', OneHotEncoder(handle_unknown='ignore'), cat_features)
])
X=full_pipeline.fit_transform(X)
```

Table of Contents

K Nearest Neighbor Regression

Goal: given $x \in \mathbb{R}^d$, predict sales.

- Find k nearest data points to x.
- Compute the predicted sales based on these k points.

```
from sklearn.neighbors import KNeighborsRegressor
model = KNeighborsRegressor(n_neighbors=k).fit(X_train,
      Y_train)
res = model.predict(X_test, Y_test)
```

K Nearest Neighbor Regression

Goal: given $x \in \mathbb{R}^d$, predict sales.

- Find k nearest data points to x.
- Compute the predicted sales based on these k points.

```
from sklearn.neighbors import KNeighborsRegressor
model = KNeighborsRegressor(n_neighbors=k).fit(X_train,
      Y_train)
res = model.predict(X_test, Y_test)
```

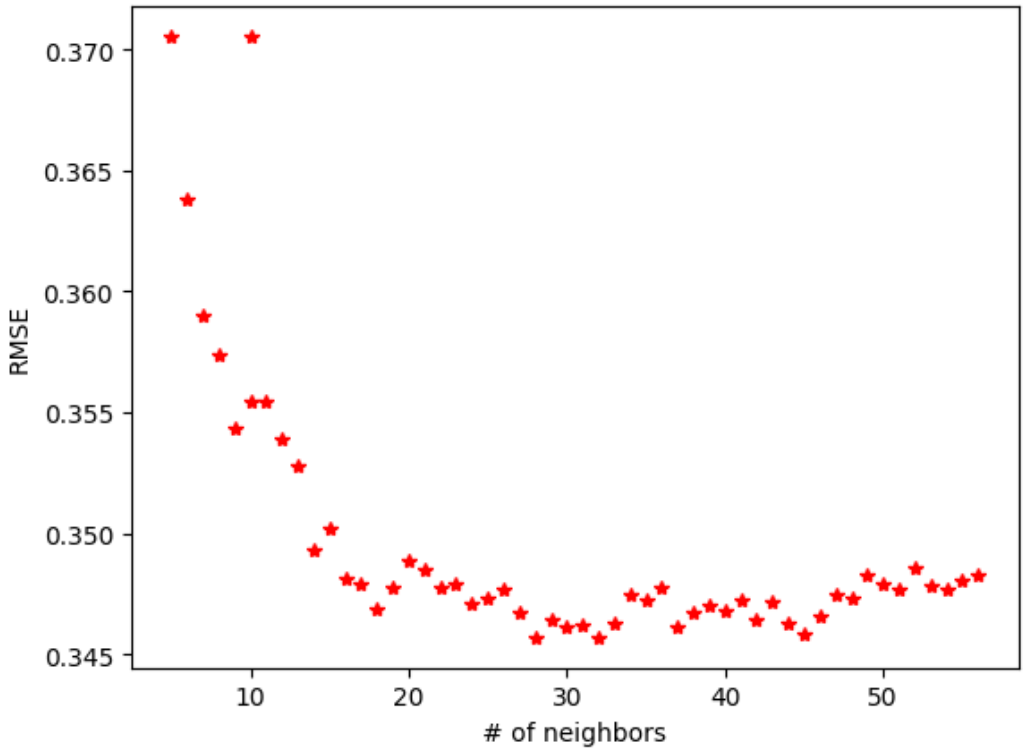
Questions we should think about:

- How to determine k?
- How to find the nearest points efficiently?
- How to predict the sales based on the points?

Cross Validation

How to determine k?

- Divide the training dataset into two parts $*$
- K-fold cross validation: divide the data into p equal parts
  - $\epsilon_p(k) = \sum_{i \in p\text{th part}} (y_i - f(x_i))^2$
  - $\epsilon(k) = \frac{1}{p} \sum_{i=1}^{p} \epsilon_p(k)$



Kd-Tree

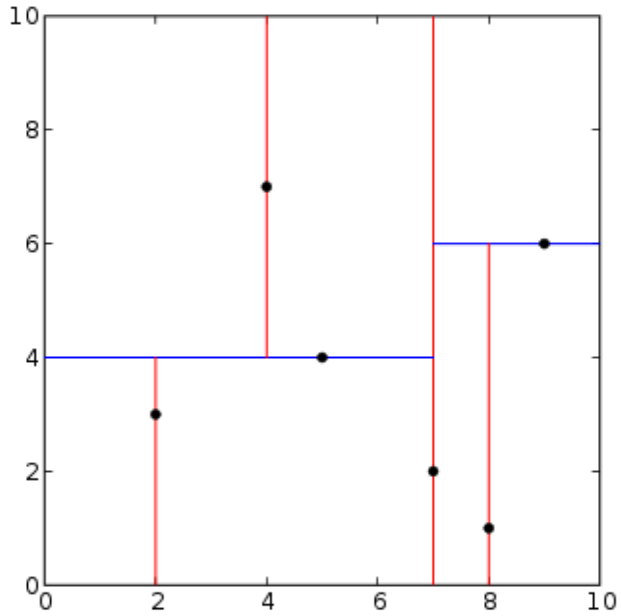
How to find the nearest points efficiently given that the training size is n with dimension d, assuming that we are using the Euclidean metric?

- Naive approach
  - Compare with all data points in the training set
  - Time Complexity: $\Theta(nd)$

Kd-Tree

How to find the nearest points efficiently given that the training size is n with dimension d, assuming that we are using the Euclidean metric?

- Naive approach
  - Compare with all data points in the training set
  - Time Complexity: $\Theta(nd)$
- Kd-Tree
  - Construct a Kd-Tree by recursively partition the plane to two halves and balancing it.
  - Search in a bushy binary tree
  - The number of features we use is small.
  - Time Complexity: $\Theta(d \log n)$.



Sales Prediction

Given $(x_1, y_1), \ldots, (x_k, y_k)$, and x, how should we predict sales based on nearest points?

- Weighted Mean
  - $d_{\text{total}} = \sum_{i=1}^{k} d(x, x_i)$
  - $y = \sum_{i=1}^{k} \frac{d(x,x_i)y_i}{d_{\text{total}}}$
- Median
- Linear Regression

Results

For our purposes, we used the K Nearest Neighbors Regressor from sklearn with Kd-Tree and weighted distance for prediction. (RMSE: 0.33)

- Pros
  - No assumptions about the data
- Cons
  - Localized data when k increases ($k = 20$ in this case).
  - Memory inefficient and slow

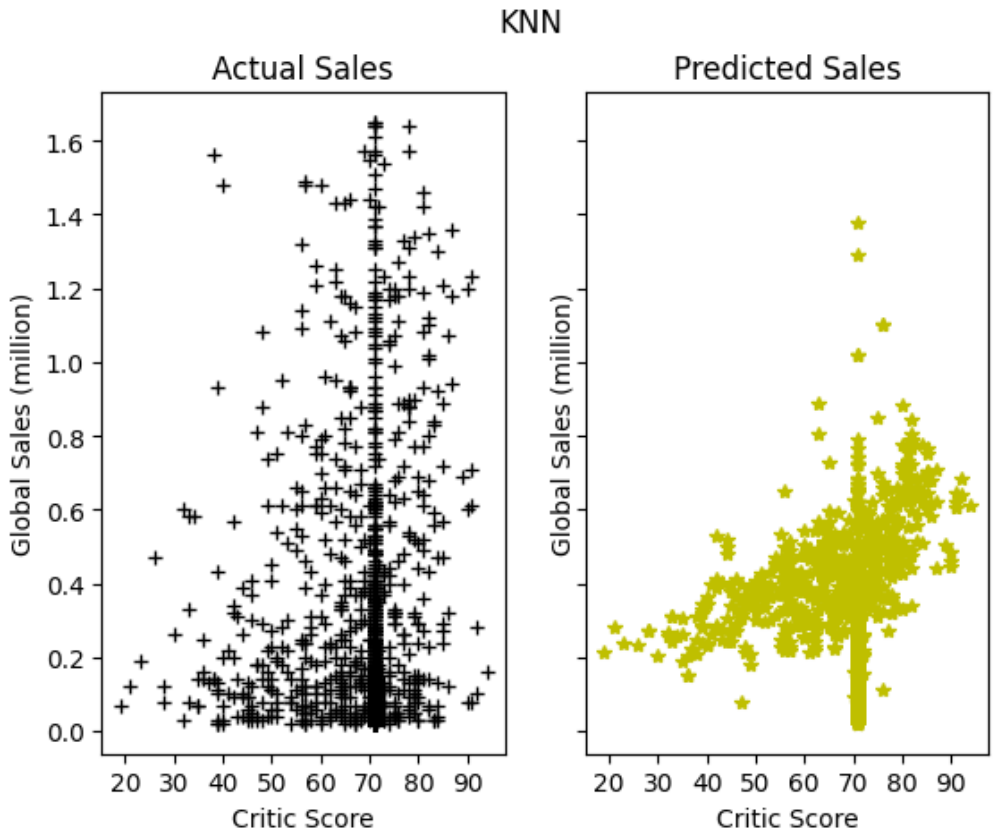


Table of Contents

Random Forest

- The Random Forest model is known as an ensemble learning method which uses multiple decision trees to be trained and in the case of regression, outputs the mean prediction of the individual trees.

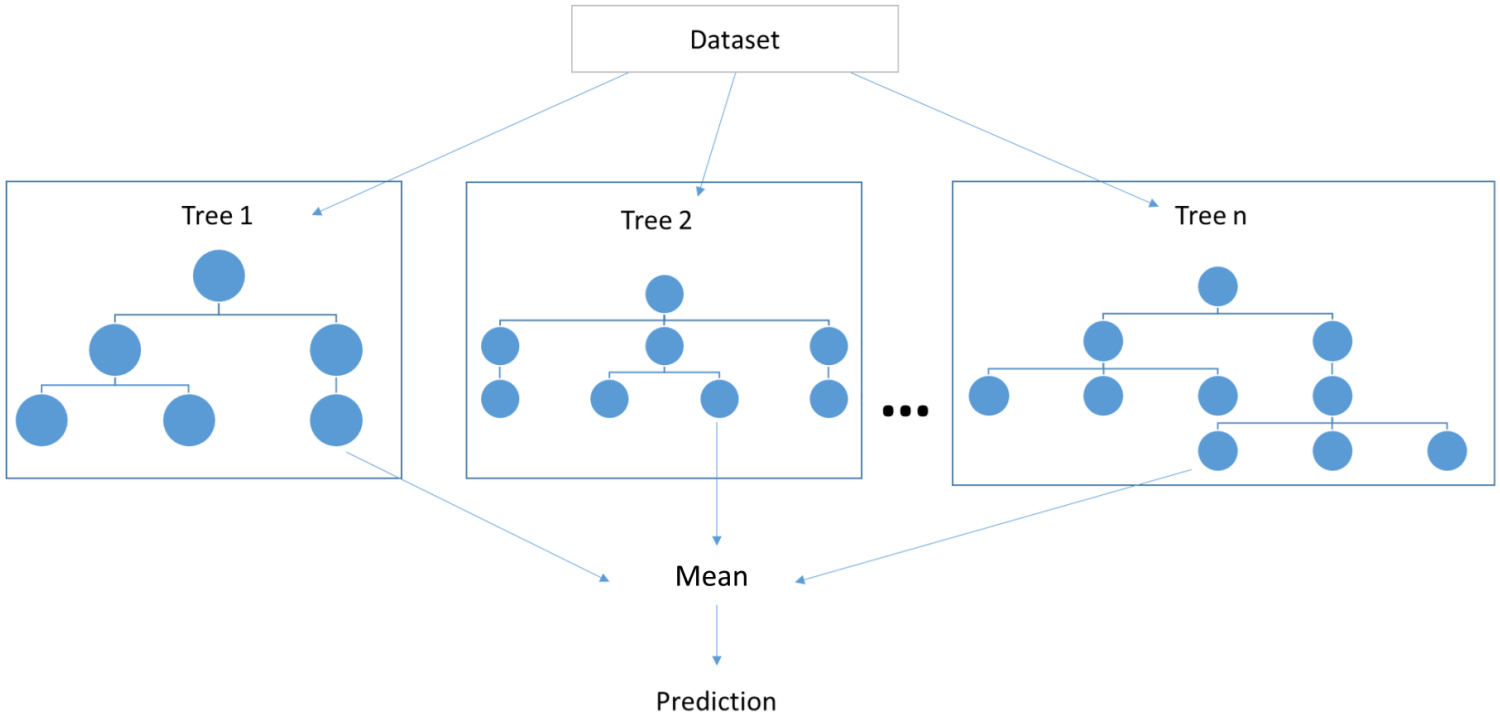


Figure: Random Forest Regressor

Random Forest

More about Random Forest:

- Random Forest differs from bagging decision trees because each of the trees are trained from a subset of the features with a process known as the Random Subspace method.
- Random Forest remedies a single decision tree's tendency to overfit the data and controls variance.
- Compared to Neural Networks, it can give good results with fewer data samples.
- Low computational cost

Random Forest

For our purposes, we used the Random Forest regressor from sklearn, which defaults to 100 trees and uses mean squared error as criterion to measure the quality of a split. (RMSE: 0.32)

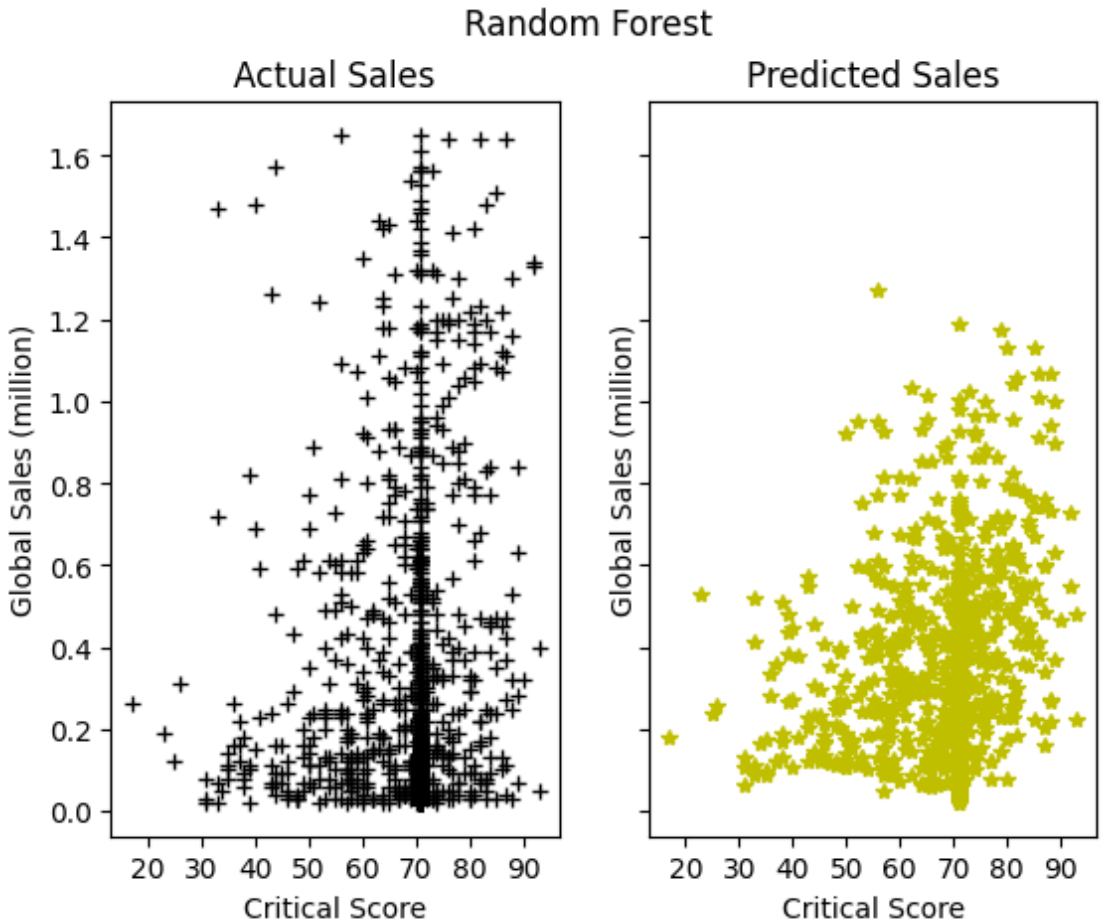


Table of Contents

Artificial Neural Network

Chen's slide.