Predicting Youtube Video Likes

Group Members: Andrew Jarmin, Cristian Moreno, Geovanny Huerta

Project Description, Goals, and Details

Project Details

- Description: Based off the Pog Champs Challenge, where members of the kaggle and twitch community combine together to create models for predicting the 'like to view_count' ratio of Youtube Videos.

- Goal: Predict the number of likes a Youtube video can receive based of the # of features we use from the dataset provided.

- Details: We will Implement this project using the Machine Learning Algorithms learned in class.

Data / Data Preprocessing

- Dataset: https://www.kaggle.com/competitions/kaggle-pog-series-s01e01/overview/evaluation

- Contains:
  - 1 Folder - ('thumbnail' images)
  - 2 Parquet Files - ('train.parquet' & 'test.parquet')
  - 1 csv file - ('sample_submission.csv)

- Our Project Dataset Details: We created 3 separate datasets (92,275 items) for our models to go through
  - Dataset 1: Consisted of 10 features, and 1 label
  - Dataset 2: Consisted of 12 features, and 1 label
  - Dataset 3: Consisted of 26 features, and 1 label

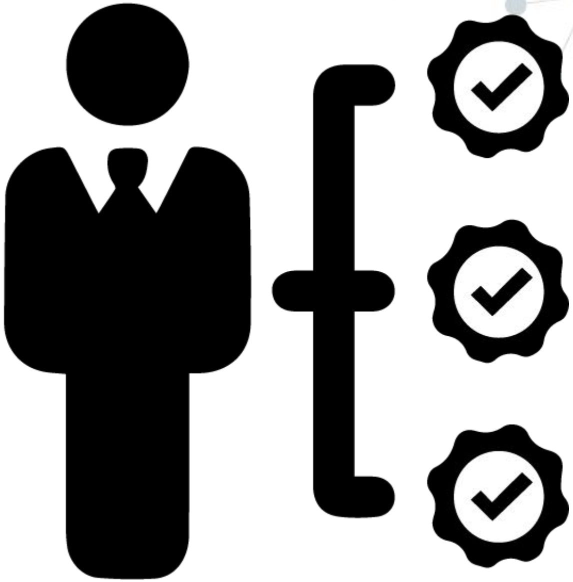
- Additional Information:
  - OHE Encoding (Trending_Day, Published_Day)
  - Boolean Values were converted to '0' and '1'
  - String Values (Titles, Channel Titles, Description) were converted to integer values by taking their lengths as their identifier)

Responsibilities

- Geovanny Huerta
  - Building the Linear SVR(Regression Model)
  - Researched the Possible Models that can be used for the Project

- Cristian Moreno
  - Data Preprocessing
  - Building the ANN-R(Regression Model)

- Andrew Jarmin
  - Building the Random Forest(Regression Model)
  - Overlooking our code



Developed Methods, Algorithms, and Tools

Tools & Imports

Language: Python 3

Tools: Jupyter Notebook, VSCode, Google Colab

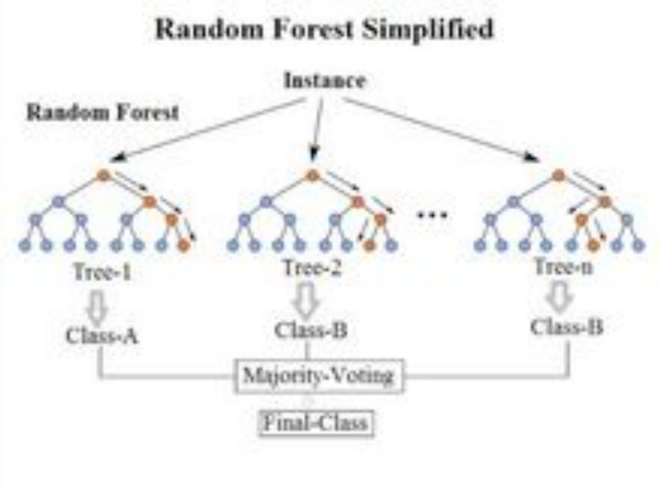
Imports: pandas, numpy, RandomForestRegressor, MLPRegressor, LinearSVR, preprocessing(for scaling), GridSearchCV, train_test_split, r2_score, mean_absolute_error, plt, tree

1. **Random Forest Regression**
2. **Linear SVR**
3. **ANN-R (Regression)**

Random Forest Regression

- **Supervised** learning algorithm that uses ensemble learning method for regression.

```
import RandomForestRegressor
rfr = RandomForestRegressor(n_estimators = 500, random_state = 0)
```



Advantage of using Random Forest Regression on this dataset:

1. runs efficiently on large datasets, and works well with both categorical and continuous variables
2. provides higher level of accuracy when predicting outcomes
3. maintains accuracy when large portions of the data are "missing"

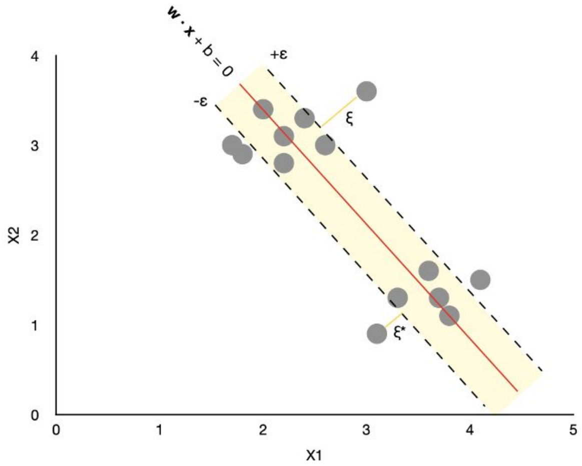
Linear SVR

- **Supervised** learning algorithm that is built based on the concept of Support Vector Machine.

```
import LinearSVR
lsvr = LinearSVR(C= 0.01)
lsvr2 = LinearSVR(C= 0.01)
lsvr3 = LinearSVR(C= 1)
lsvr4 = LinearSVR(C= 1)
```

Advantage of using Linear-Support Vector Regression on this dataset:

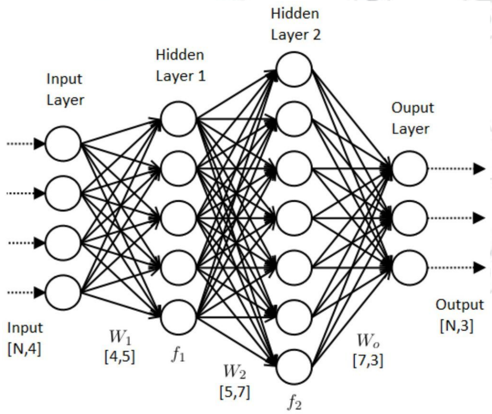
1. excellent generalization capability, with high prediction accuracy
2. very robust to outliers
3. decision model can be easily updated



*Although not suitable for large datasets. But if needed then Linear SVR provides faster results opposed to RBF SVR.

ANN-R (Regression)

- **Supervised** learning algorithm that based on the concept of “neurons” similar to the human brain.




```
import MLPRegressor
ANN = MLPRegressor(random_state=4662, hidden_layer_sizes=(85, 60), learning_rate_init=0.1, max_iter=500, verbose=False)
```

Advantage of using Artificial Neural Network - R on this dataset:

1. ANNs can generalize and predict on unseen data
2. Ability to learn and model non-linear and complex relationships
3. Can perform multiple task in parallel without affecting the system performance (works well with large datasets)

Hidden_layer_sizes: tuple
- The ith element represents the number of neurons in the ith hidden layer.

Learning_rate_init : float
- learning rate used, controls the step-size in updating the weights

Max_iter : int
- Maximum number of iterations

Problems We Encountered

- Which features to keep/remove
    - Too many non-numerical data [8 string type cols | 2 date type cols]
    - 3 not included in testing set
    - Left with 7 out of 20 features in our main dataset

- Which features to add?
    - Added 4 to main, 6 more to opt1, and 20 more to opt2
    - Total: 11 in Main, 13 in Opt1, 27 in Opt2

- NaN errors in results
    - Error didn't mention which column
    - Turned out it was duration_seconds with NaN data

- Testing set did not have the actual values
    - No way of verifying the testing set results
    - Trained our models based on the testing set columns

ML Algorithm’s Performance (Best Model Results For Each)

```
Main Dataset - R_Squared w/ n_estimators 10, 50, 100
---------------------------------------------------
R_Squared = 0.834 %
R_Squared = 0.869 %
R_Squared = 0.875 %
Main Dataset - MAE w/ n_estimators 10, 50, 100
---------------------------------------------------
MAE = 43898.953
MAE = 40532.991
MAE = 39682.366
```

← Random Forest Regressor Results

- Dataset 1

```
    0.779 : R^2 (best possible score is 1.0)
90398.757 : Mean Absolute Error
```

ANN-Regression Results ->

- Dataset 2

```
    -0.011 : R^2 (The best possible score is 1.0, lower values are worse.)
119789.711 : Mean Absolute Error
```

← Linear SVR Results

- Dataset 3

Conclusion

- After Evaluating all three Algorithms we've seen that Random Forest was the best model.
- Linear SVR had the worst model results.
- OHE proved to be a detriment for our Project
- Best Accuracy: 0.88% using n_estimators = 100
- Lowest MAE Score: 39,682 using n_estimators = 100

```
Main Dataset - R_Squared w/ n_estimators 10, 50, 100
----------------------------------------------------
R_Squared = 0.834 %
R_Squared = 0.869 %
R_Squared = 0.875 %
Main Dataset - MAE w/ n_estimators 10, 50, 100
----------------------------------------------------
MAE = 43898.953
MAE = 40532.991
MAE = 39682.366
```