Predicting Global Sales of Video Games

Kevin Weng, Yi Lyu, Chen Li, Omar Hafez
MATH 156 Final Project: Group 5
University of California, Los Angeles (UCLA)
Github Project

Abstract

Our project aims to model global video game sales by applying a neural network, Random Forest, and k-Nearest Neighbors model to preprocessed data to successfully predict global sales and determine which model works best. We built the project on Python using sklearn, Keras, and Tensorflow for the models and SQL for data processing. We will evaluate the results by root mean squared error (RMSE). We hope to see whether or not global sales can be effectively modeled and also give an analysis on model strengths and weaknesses for this task.

1 Introduction

The expansion of the video game industry and eSports in the past decade has fueled gamers and game studios all over the world. With the backdrop of COVID-19, worldwide quarantines, and work-from-home structures, video games have garnered another boost in popularity and success in 2020. With companies like Nintendo, Activision Blizzard, and many independent developers making an impact on modern culture, the landscape of video games and entertainment software has drastically changed. Creating a video game is an intensive project that requires a diverse array of resources and specialists that would be very costly for the studio if a release goes awry. For video game producers, investors, and consultants, the ability to project global sales is very useful when it comes to considering possible translations, global releases, and general marketing investment in other parts of the world.

In this work, we apply artificial neural network, Random Forest, and k-Nearest Neighbors to model global video game sales in Section [2], select the optimal prediction model in Section [5.3], and discuss the predictability of video game sales based on our data in Section [6]. Before describing the sales prediction models, we first provide an overview of related works in Section [2] and the preprocessing steps taken on the dataset in Section [3.2].

2 Related Works

There are other works related to our project in the video game industry, and countless more in the generalized sales and marketing prediction field. One of the projects involves internet search volume as a feature to predict global sale and is becoming increasingly relevant as social media dominates the information space in the majority of the video game industry's target audience[5]. This data is likely heavily correlated to global sales as the consumer sentiment is captured even prior to release. Another paper that used neural networks predicted weekly game sales on PCA preprocessed data[3]. The weekly timeframe is different from our cumulative global sales number and may be impacted seasonally. Additionally, one other paper used sexualized cover art content as a feature to predict sales for video games[4]. In this case, another specific feature was analyzed that we were not able to consider for our project. As pertaining to sales, features regarding behavioral economics, consumer psychology, and marketing can all be possible candidates to more successfully model video game sales.

3 Dataset and Features

The training data set is Video Game Sales with Ratings from Kaggle. The dataset consists of $11,563$ video game titles detailing release year, publisher, platform, genre, regional sales, global sales, critic and user scores, critic and user counts, and ESRB rating. Not all features are present for every title. The critic and user scores were obtained from Metacritic, a popular video game review site.

	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 1: First 5 entries of dataset

3.1 Features

Name - The name of the video game.

Platform - The console on which the game runs on. (Wii, PS4, PC, etc.)

Year of Release - The year the game was released.

Genre - Category of the game. (Shooter, Racing, Puzzle, etc.)

Publisher - Publisher of the game.

NASales, EUSales, JPSales, OtherSales - regional sales of video gamesin millions of units.

Global Sales - Total sales in the world in millions of units on a particular platform.

Critic score - Score by Metacritic's critics.

Critic count - Number of critics who contributed to Critic_score.

User score - Score by Metacritic's subscribers.

User count - Number of users who contributed to User_score.

Developer - Party who created the game.

Rating - The Entertainment Software Rating Board (ESRB) rating.

3.2 Preprocessing

First, we removed the games missing **Platform, Genre, Publisher, and Year of Release** data in that these variables could not be imputed effectively. We then imputed with median **Critic score, User score, Critic count, and User count** because many are missing values. For the remaining categorical data in **Genre and Publisher**, we used one-hot-encoding to make the data suitable for regression and dropped one column of each to decorrelate the columns. Our preprocessing also removed the local sales of games because they were often obtained after global sales were calculated and thereby would not be useful for showing a correlation with prior regional video game sales and a global launch. Because of the uncommon popularity of certain games, such as Grand Auto Theft V, we consider the top 10% and bottom 10% as outliers and remove them accordingly. Also, games published before 2004 are not considered due to a smaller size of the gaming industry at that time; the year 2004 is chosen for the release of War of Warcraft (WoW). In addition, we group video games by their **names** so that the global sales data do not concern the platforms these games are/were published on.

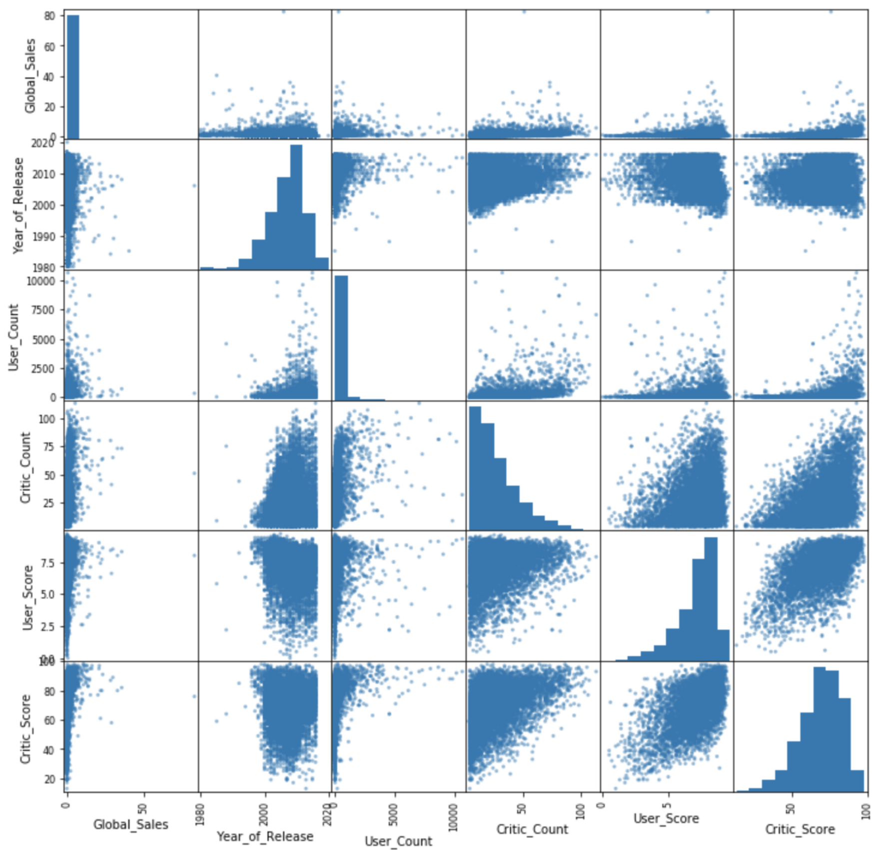

Figure 2: Correlation Matrix of Features (Partial)

After preprocessing the data, we split the remaining into 80% training data and 20% test data. For k-Nearest Neighbors, we use cross validation to determine the optimal number of nearest neighbors. Two approaches have been attempted: we split the data into three categories: training, test, and cross validation, which is then used to determine the optimal k number of neighbors that minimizes RMSE; we apply k-fold cross validation to the dataset, which will be discussed in detail in Section 4.1. The correlation matrix in Figure 2 is used to determine whether or not any features should be dropped. After feature evaluation, we find that **critic score, user score, genre, and publisher** potentially have a great impact on video games sales prediction and are consequently chosen to be the features we use for training.

4 Methods

4.1 Models

Neural Network - The Neural Network, as studied in class, is a model comprised of hidden layers of nodes known as neurons that takes in an input and, by feeding it through the hidden layers, produces a result in the output layer. In the Neural Network model we use, we apply the state-of-the-art optimizer NAdam[7] to our model and use the rectified linear unit (ReLU) as the activation function in that functions like $\tanh$ and sigmoid are hard to train given the limited size of our dataset, and ReLU, by definition, does not permit negative values, which coincides with our purpose of predicting global sales.

Random Forest - The Random Forest model is in ensemble method that trains multiple decision trees and outputs a class by majority vote[1]. For regression tasks, instead of mode, the mean prediction of the individual trees is returned. For reference, a decision tree is a popular machine learning algorithm that uses many input variables to traverse down a tree, and returns a prediction from a leaf. The benefit of using multiple trees is a reduced variance as a single tree can easily overfit the data. Random Forest differs from simply bagging multiple decision trees by selecting a random subset of the features for each tree so that if some features are stronger predictors than others, such trees would be correlated known as the 'Random Subspace method[6].'

k-Nearest Neighbors - The k-Nearest Neighbors algorithm takes an element and looks for its closest neighbors to take a majority vote in the classification case, and the mean or median value of the nearest neighbors for regression[2]. Mean is chosen in our case. The basic structure of the algorithm is described as follows: for all possible k for our model,

1. We divide the training dataset into p equal parts, where p is fixed.
2. We randomly choose one part for cross validation and the remaining $p-1$ parts for training, which we will repeat for p times so that each part is used once as cross validation set, yielding us p errors. We then compute the average error $\epsilon(k)$ of this model given k over the p parts.
3. We find the k that minimizes the average error $\epsilon(k)$ and return the model.

For our experiment, we weighed the points by the inverse of their distance, making closer neighbors have greater influence: $\hat{y} = \sum_{i=1}^{k} \frac{d(x,x_i)y_i}{\sum_{i=1}^{k} d(x,x_i)}$

5 Experiments/Results/Discussion

5.1 Methodology

After training the three models, we will evaluate the results with the root mean squared error (RMSE) metric to evaluate efficacy and accuracy.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

where y_i is the global sales corresponding to the test input x_i, and $\hat{y}_i$ is the prediction of our models. RMSE takes into account negative values and is a commonly used metric in determining the performance of regression models.

5.2 Results

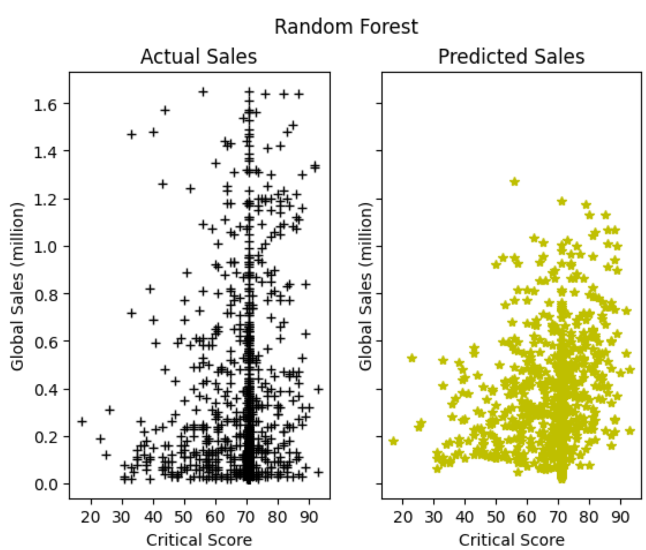

Figure 3: Random Forest Predictions

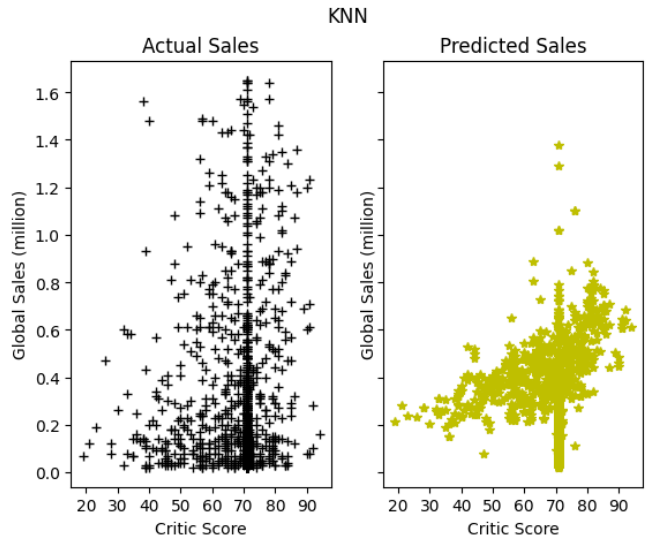

Figure 4: k-Nearest Neighbors Predictions

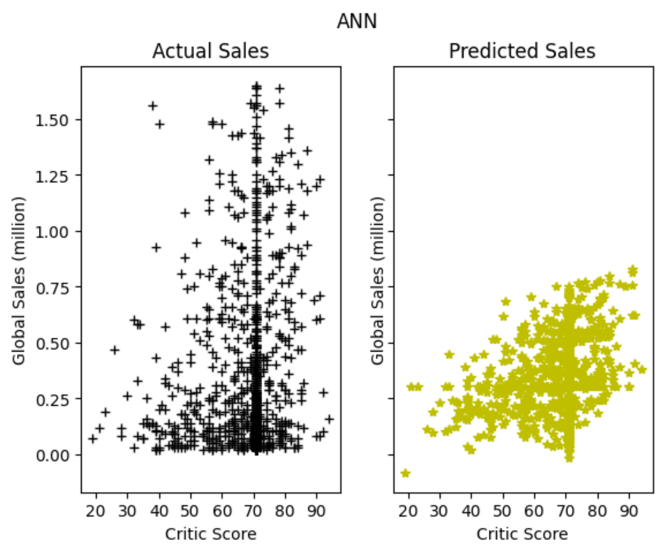

Figure 5: Artificial Neural Network Predictions

As we can see in Figure 3, Figure 4, and Figure 5[1], the predicted sales roughly capture the general distribution of the global sales, but the one corresponding to k nearest neighbors regression is more localized than are the other twos, which can be ascribed to the fact that the k nearest neighbors regression only looks at the neighbors of the input for prediction, while the other two consider the data globally. Although Artificial Neural Network appears better than KNN in predicting global sales, it returns a negative global sales, which is undesirable. Through computation, we can have

$\mathrm{RMSE}_{\mathrm{RF}}$:	0.3216
$\mathrm{RMSE}_{\mathrm{kNN}}$:	0.3315
$\mathrm{RMSE}_{\mathrm{ANN}}$:	0.3562

[1]The labels of x axis in Figure 3, Figure 4, and Figure 5 above are misleading because features other than critic score have been used in our models; we consider scatter plots are better in terms of visualization.

5.3 Discussion

From our RMSE values, we see that the strongest performer was Random Forest and the weakest was the ANN trained with 10 epochs. Increasing the number of epochs to 200 improved the RMSE to 0.3319, still the worst performing model. Naturally, we will discuss the strengths and weaknesses of our models. For Random Forest, an ensemble of regression trees, the individual trees are easy to understand, specifically for mirroring human behavior purposes which is relevant to our task of suggesting that features impact decisions to purchase video games, but because of the nature of Random Forest, modelers have few controls over it. In contrast with Random Forest, Neural Network is incomprehensible, making it less attractive for practical uses of sales prediction. However, as mentioned previously, there have been neural networks applied to PCA processed data for sales prediction. Regarding the k-Nearest Neighbors algorithm, as mentioned in Section 5.2 above, it falls in between the other two models in terms of RMSE, which implies that the model performs relatively well on this dataset, but the predicted sales for k-Nearest Neighbors regression are localized, as shown in Figure 4, which leads to a low variance. In addition, that k-Nearest Neighbors regression has to store all the data in the training set renders it inefficient when the size of the data set increases.

6 Conclusion and Future Work

With an RMSE of around 0.33, our objective of modeling global sales of video games is decently accurate and can provide some insight for future video game releases, especially considering that the range of global sales value is 0 to 60. Also, the models have a low chance of overfitting because ensemble methods and cross validation, which are used in our models, are innately preventative measures against overfitting. Out of the regressors that we explored, we conclude that the Random Forest model produced the best prediction of global video game sales by measure of RMSE, albeit by a small margin. To continue our exploration in the realm of video games, we see that some of the most successful and popular games today are free to play, offering paid in-game content that users can elect to pay for or not. Our project does not strongly consider this model of games and exploration of different payment structures may be interesting to consider in the future. Plus, because we exclude those extremely popular and unpopular as outliers, our models can only predict the sales of normal video games. Furthermore, as seen in related work, there are many esoteric features that were not taken into account. As with any consumer product, sentiment is a major factor in how a product is received and could also be a viable direction of exploration. Separate analyses of games and marketing strategies can be helpful to building a more holistic and complete model of global sales.

References

[1] Anava, Oren, and Kfir Levy. ”k*-nearest neighbors: From global to local.” Advances in neural information processing systems. 2016.

[2] Breiman, Leo. ”Random forests.” Machine learning 45.1 (2001): 5-32.

[3] Marcoux, Julie, and Sid-Ahmed Selouani. ”A hybrid subspace-connectionist data mining approach for sales forecasting in the video game industry.” 2009 WRI World Congress on Computer Science and Information Engineering. Vol. 5. IEEE, 2009.

[4] Near, Christopher E. ”Selling gender: Associations of box art representation of female characters with sales for teen-and mature-rated video games.” Sex roles 68.3 (2013): 252-269.

[5] Ruohonen, Jukka, and Sami Hyrynsalmi. ”Evaluating the use of internet search volumes for time series modeling of sales in the video game industry.” Electronic Markets 27.4 (2017): 351-370.

[6] Wikipedia contributors. ”Random subspace method.” Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 3 Nov. 2019. Web. 27 Jul. 2020.

[7] Dozat, Timothy. ”Incorporating Nesterov Momentum into Adam.” ICLR 2016 workshop. 2016.

```python
"""
================================================
==  Filename: helper.py
==  Author: Yi Lyu
==  Status: Complete
================================================
"""

import numpy as np
import pandas as pd
import pickle
import sqlite3
import re
import os
from sklearn.impute import SimpleImputer as Imputer
from sklearn.impute import KNNImputer
from sklearn.preprocessing import LabelEncoder

__all__ = ['Videogames']

def get_dir(path):
    return os.path.join(getWorkDir(), path)

def getWorkDir():
    pathlist = os.path.abspath(os.curdir).split('/')
    path = '/'
    for p in pathlist:
        path = os.path.join(path, p)
        if p == 'video-game-sales-predictor' or p == 'video-game-sales-predictor-master':
            break
    return path

class Videogames(object):
    def __init__(self, database_dir, data_dir='data/', storage='data'):
        self.database_dir = database_dir
        self.table = ''
        self.data_dir = data_dir
        self.storage = '{0}.pickle'.format(storage)

        self._has_data = False
        self._headers = []
        self._dtypes = []
        self._connection = None
        try:
            with open(get_dir(data_dir + self.storage), "rb") as f:
                self.table, self._headers, self._dtypes, self._has_data = pickle.load(f)
        except:
            pass

    @property
    def table_name(self):
        return self.table

    @property
    def status(self):
        return self.get_status()

    @property
    def headers(self):
        return self._headers

    @property
    def dtypes(self):
```

```
        return self._dtypes

    def get_status(self):
        return self._has_data

    def get_headers(self):
        return self._headers

    def get_dtypes(self):
        return self._dtypes

    def read_data_in(self, filepath, table, write_headers=False):
        conn = sqlite3.connect(database=self.database_dir)
        cur = conn.cursor()
        data = pd.read_csv(filepath, delimiter=",", encoding="unicode_escape")

        self.table = table
        headers = self._get_headers(data)
        dtypes = self._get_dtypes(data)
        self._create_table(headers, dtypes, cur)

        if write_headers:
            with open(get_dir(self.data_dir + 'headers.csv'), "w+") as f:
                f.write(", \n".join(headers))

        if not self._has_data:
            data = self._remove_missing(data)
            with open(get_dir(self.data_dir + self.storage), "wb+") as f:
                self._insert_data(data, headers, dtypes, cur)
                self._has_data = True
                pickle.dump((self.table, headers, dtypes, True), f, pickle.HIGHEST_PROTOCOL)

        del data
        conn.commit()
        conn.close()

    def _remove_missing(self, data):
        data = data.replace(r'tbd', np.nan, regex=True)
        data['User_Score'] = data['User_Score'].astype(np.float64)
        condition = (data['Platform'].notnull() & data['Genre'].notnull() & data['Publisher'].notnull() & data['Year_of_Release'].notnull())
        data = self._imputation(data[condition])
        return data

    def _imputation(self, data):
        imp = Imputer(strategy='median')
        attributes = ['Critic_Score', 'User_Score', 'Critic_Count', 'User_Count']
        for item in attributes:
            data[item] = imp.fit_transform(data[[item]]).ravel()
        return data

    def get_col(self, *header):
        if not self._connection:
            self._connection = sqlite3.connect(self.database_dir)
        cur = self._connection.cursor()

        command = "SELECT {0} FROM {1};".format(self._list2str(header), self.table)
        return self._col2list(cur.execute(command).fetchall())

    def execute(self, command):
        if not self._connection:
```

```python
            self._connection = sqlite3.connect(self.database_dir)
        cur = self._connection

        if bool(re.match("^[ \t\n]*SELECT", command, re.I)):
            return list(self._col2list(cur.execute(command).fetchall()))
        else:
            print("ILLEGAL COMMAND")


    ## Helper Functions ##
    def _get_headers(self, data):
        """Return the headers of the data

        Args:
            data DataFrame: the data we read from csv.

        Returns:
            list: the headers of the data
        """
        if not self._headers:
            self._headers = list(map(lambda col: col.lower(), data.columns))
        return self._headers

    def _get_dtypes(self, data):
        if not self._dtypes:
            self._dtypes = [self._process_dtype(data[col][0]) for col in data.columns]
        return self._dtypes

    def _create_table(self, headers, dtypes, cur):
        """Execute the following SQL command

        CREATE TABLE IF NOT EXISTS {table} (
            name VARCHAR(80),
            ...
        );

        Args:
            headers (list): the list of columns where each header is lowercase.
            dtypes (list): the list of types where each type is either NUMBER or VARCHAR(80) based on this data set.
            cur (sqlite3.connection.cursor): a connection cursor of sqlite3 database
        """
        command = "CREATE TABLE IF NOT EXISTS {0} (".format(self.table)
        template = "{0} {1}"

        n = len(headers)

        ## Convert the data to suitable form for _list2str function
        data = [template.format(headers[i], dtypes[i]) for i in range(n)]

        command += self._list2str(data)
        command += ");"
        cur.execute(command)

    def _insert_data(self, data, headers, dtypes, cur):
        command_template = "INSERT INTO {0} ({1}) VALUES ({2});"
        for i, itr in data.iterrows():
            res = list(map(self._str_classifier, list(itr)))
            command = command_template.format(self.table, ", ".join(headers),
```

```
                                                            self._list2str(res,
    classify=self._row_classifier(res, dtypes)))
            cur.execute(command)

    def _list2str(self, data, delimiter=",", classify=lambda x: x):
        """Convert the list to a string

        I have not found such a function in Python and therefore
        wrote one.

        Args:
            data (list): the row of the table
            delimiter (str, optional): the delimiter.
            classify (function, optional): a function that classifies the
     data in the row.

        Returns:
            str: a string representing the data converted to a string.
        """
        res = ""
        for i in range(len(data)):
            res += classify(data[i])
            if i != len(data) - 1:
                res += delimiter + " "
        return res

    def _row_classifier(self, data, dtypes):
        ### classify the data in a row in the table
        def classifier(x):
            i = data.index(x)
            if dtypes[i] == "NUMBER":
                if x == "NULL" or x == 'tbd':
                    return "-1"
                else:
                    return str(x)
            else:
                return "\"{0}\"".format(x)
        return classifier

    def _str_classifier(self, x):
        ### classify the data so that it does not contain nan
        if type(x) == float and np.isnan(x):
            return -1
        return x

    def _process_dtype(self, var):
        dtype = type(var)
        if dtype == str and var.isnumeric():
            return "NUMBER"
        type_converter = {type(',') : "VARCHAR(80)", np.float64: "NUMBER"
    , np.int64: "NUMBER"}
        return type_converter[dtype]

    def _col2list(self, col):
        n = len(col[0])
        return list(map(lambda x: list(x)[:n], col))
```

```
"""
================================================
==  Filename: main.py
==  Author: Yi Lyu
==  Status: Complete
================================================
"""

```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import pickle
import os
from sklearn.decomposition import PCA
from keras.models import load_model
from sklearn.model_selection import train_test_split

from helper import Videogames, getWorkDir, get_dir
from models import *
from plotting import *

def read_data():
    videogames = Videogames(get_dir("data/math156.db"))
    videogames.read_data_in(get_dir("data/videogames.csv"), "VIDEOGAMES", True)
    res = np.array(videogames.execute('''
        SELECT name, g_total, cscore, uscore, genre, publisher FROM (
            SELECT name AS name,
                   SUM(global_sales) AS g_total,
                   critic_score AS cscore,
                   user_score AS uscore,
                   genre AS genre,
                   publisher AS publisher
            FROM VIDEOGAMES
            WHERE year_of_release >= 2004 and uscore != 0 and cscore != 0
            GROUP BY name) AS VideogameSummary
        WHERE g_total != 0
        ORDER BY g_total DESC;
        '''))
    return res

if __name__ == "__main__":
    ## the critic scores and user scores
    columns = ['name', 'gtotal', 'cscore', 'uscore', 'genre', 'publisher']
    res = pd.DataFrame(read_data(), columns=columns)

    n = len(res)
    factor = 0.1
    quantile1 = round(n * factor)
    quantile2 = n - round(n * factor)
    res = res.loc[quantile1:quantile2 + 1, :]

    ## Transform data into appropriate form for regression
    scores = res[['cscore', 'uscore']]
    genre = pd.get_dummies(res['genre'], drop_first=True)
    publisher = pd.get_dummies(res['publisher'], drop_first=True)

    X = pd.concat((scores, genre, publisher), axis=1).astype('float64')
    Y = res['gtotal'].astype('float64')


    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=.20, train_size=.80, random_state = 40)

    try:
        with open(get_dir('data/models.pickle'), 'rb') as f:
            rfregr, knnregr = pickle.load(f)
            annregr = load_model(get_dir('data/ann'))
    except:
        annregr = ann(X_train, Y_train.ravel())            ## ANN
```

```
        rfregr = random_forest(X_train, Y_train.ravel())    ## Random
    Forest
        knnregr = knn(X_train, Y_train.ravel())             ## KNN
        with open(get_dir('data/models.pickle'), 'wb+') as f:
            pickle.dump((rfregr, knnregr), f, pickle.HIGHEST_PROTOCOL)
            annregr.save(get_dir('data/ann'))

    print("The mean is", np.mean(Y))
    ## RMSE
    rmse_template='RMSE\t{name:25}{value:18}'
    print(rmse_template.format(name='random forest', value=rmse(X_test,
    Y_test, rfregr)))
    print(rmse_template.format(name='Knn', value=rmse(X_test, Y_test,
    knnregr)))
    print(rmse_template.format(name='ANN', value=rmse(X_test, Y_test,
    annregr)))

    plot_predictions(X_test, Y_test, rfregr, knnregr, annregr)


    """

    print('=====================================')

    ## R2
    r2_template = 'R^2\t{name:25}{value:18}'
    print(r2_template.format(name='random forest', value=rfregr.score(
    X_test, Y_test)))
    print(r2_template.format(name='Knn', value=knnregr.score(X_test,
    Y_test)))
    print(r2_template.format(name='Aan', value=annregr.score(X_test,
    Y_test)))

    """
```

```
"""
================================================
==  Filename: models.py
==  Author: Yi Lyu
==  Status: Complete
================================================
"""

import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import Ridge
from sklearn.linear_model import GammaRegressor
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense

## just in case someone wants to implement them instead of using sklearn

def rmse(X_test, Y_test, model):
    Y_pred = model.predict(X_test)
    return mean_squared_error(Y_test, Y_pred, squared=False)

def plot_knn(ns, rmses):
    plt.plot(ns, rmses, 'r*')
    plt.xlabel('# of neighbors')
```

```
    plt.ylabel('RMSE')

    plt.savefig('graphs/knn_choice_n.png', bbox_inches='tight')
    plt.clf()

def knn(xs, ys, n=10):
    #X_train, X_test, Y_train, Y_test = train_test_split(xs, ys, test_size= .1, random_state = 40)
    num_cols = len(xs.columns)
    i = 5

    best_index = 4
    best_score = 10000
    nums = [i for i in range(5, int(np.sqrt(num_cols)) + 10)]
    cvs = []

    for num in nums:
        model = KNeighborsRegressor(n_neighbors=num, algorithm='kd_tree', weights='distance')
        temp = cross_val_score(model, xs, ys, cv=5).mean()
        temp = np.sqrt(1 - temp)
        if temp < best_score:
            best_score = temp
            best_index = num
    print(best_index)
    return KNeighborsRegressor(n_neighbors=best_index, algorithm='kd_tree', weights='distance').fit(xs, ys)
    """

    best_model = KNeighborsRegressor(n_neighbors=i, algorithm='kd_tree', weights='distance').fit(X_train, Y_train)
    best_rmse = rmse(X_test, Y_test, best_model)

    ### Cross Validation
    ns = [n]
    rmses = [best_rmse]
    cvs = []
    ### You can change 5 to * 2 or * 3 here for a better result, but slower.
    for n in range(i, int(np.sqrt(num_cols)) + 5):
        model = KNeighborsRegressor(n_neighbors=n, algorithm='kd_tree', weights='distance').fit(X_train, Y_train)
        temp = rmse(X_test, Y_test, model)
        ns.append(n)
        rmses.append(temp)
        if temp < best_rmse:
            best_model = model
            best_rmse = temp
    plot_knn(ns, rmses)

    """

    return best_model

def ann(xs, ys):
    n = len(xs.columns)
    ANN = Sequential()
    ANN.add(Dense(units = 6, activation = "relu", input_dim = n))
    ANN.add(Dense(units = 4, activation = "relu"))
    ANN.add(Dense(units = 1))

    ANN.compile(optimizer = "adam", loss = "mean_squared_error")
    ANN.fit(xs, ys, batch_size = 2, epochs = 200)
    return ANN

```

```python
def gamma_model(xs, ys):
    model = GammaRegressor().fit(xs, ys)
    return model

def linear_model(xs, ys, m):
    model = make_pipeline(PolynomialFeatures(m), Ridge(normalize=True)).fit(xs, ys)
    return model

def random_forest(xs, ys):
    model = RandomForestRegressor(criterion='mse').fit(xs, ys)
    return model
```

```python
"""
================================================
==  Filename: plotting.py
==  Author: Yi Lyu
==  Status: Complete
================================================
"""

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

from helper import getWorkDir, get_dir

def predict(X_test, Y_test, model):
    """Predict the sales based on the dataset

    Args:
        X_test (DataFrame): Data
        Y_test (Series): Actual Sales
        model (object): Model we are using

    Returns:
        DataFrame: predicted scales
    """
    return pd.DataFrame(model.predict(X_test))

def plot_helper(xs, data_ys, predict_ys, model_name='Unknown'):
    """Plot the predicted sales

    Args:
        xs (Series): the x values
        data_ys (Series): the actual sales
        predict_ys (Series): the predicted sales
        model_name (str, optional): the name of the model. Defaults to 'Unknown'.
    """
    xs = xs.astype(np.float64)
    fig, (ax1, ax2) = plt.subplots(1, 2, sharex=True, sharey=True)
    plt.xticks(np.linspace(0, 100, 11))

    fig.suptitle(model_name)

    ax1.plot(xs, data_ys, 'k+', label='data')
    ax1.set_title('Actual Sales')
    ax1.set(xlabel='Critic Score', ylabel='Global Sales (million)')

    ax2.plot(xs, predict_ys, 'y*', label='prediction')
    ax2.set_title('Predicted Sales')
    ax2.set(xlabel='Critic Score', ylabel='Global Sales (million)')
```

```python

    pic_path = 'graphs/{0}.png'.format(model_name.replace(' ', '_').lower())

    pic_dir = get_dir(pic_path)

    plt.savefig(pic_dir, bbox_inches='tight')
    plt.clf()

def plot_helper2(data_ys, predicted_ys, model_name='Unknown'):
    fig, (ax1, ax2) = plt.subplots(1, 2, sharex=True, sharey=True)
    fig.suptitle(model_name)

    bins = np.arange(0, 6, 0.1)
    sns.distplot(data_ys, bins=bins, hist=True, kde=True, ax=ax1, color='r', axlabel='Sales')
    ax1.set_title('Actual Sales -- Density Plot')
    ax1.set_xlim(0, 2)
    sns.distplot(predicted_ys, bins=bins, hist=True, kde=True, ax=ax2, color='b', axlabel='Sales')
    ax2.set_title('Predicted Sales -- Density Plot')
    ax2.set_xlim(0, 2)

    pic_path = 'graphs/{0}_hist.png'.format(model_name.replace(' ', '_').lower())
    pic_dir = get_dir(pic_path)

    plt.savefig(pic_dir, bbox_inches='tight')
    plt.clf()

def plot_predictions(X_test, Y_test, rfregr, knnregr, annregr):
    """Plot the Predicted sales of each model

    Args:
        X_test (DataFrame): data
        Y_test (Series): actual sales
        rfregr (RandomForestRegressor): Random Forest Regressor
        knnregr (KNNRegressor): KNN Regressor
        annregr (ANNRegressor): Artificial Neural Network Regressor
    """
    cscores = X_test['cscore']
    ## Get predicted sales
    rfres = predict(X_test, Y_test, rfregr)
    knnres = predict(X_test, Y_test, knnregr)
    annres = predict(X_test, Y_test, annregr)

    ## Correct the indices in case
    temp = pd.DataFrame(pd.concat([cscores, Y_test], axis=1).to_numpy(),
                        columns=['cscore', 'gtotal'],
                        index=np.arange(0, len(cscores), 1))

    ## Create a pandas DataFrame sorted by Critic Score
    df = pd.concat([temp, rfres, knnres, annres], axis=1)
    df = pd.DataFrame(df.sort_values(by='cscore', ascending=True).to_numpy(),
                 columns=['cscore', 'gtotal', 'rfres', 'knnres', 'annres'])

    plot_helper(df['cscore'], df['gtotal'], df['rfres'], 'Random Forest')
    plot_helper(df['cscore'], df['gtotal'], df['knnres'], 'KNN')
    plot_helper(df['cscore'], df['gtotal'], df['annres'], 'ANN')

    plot_helper2(df['gtotal'], df['rfres'], 'Random Forest')
    plot_helper2(df['gtotal'], df['knnres'], 'KNN')
    plot_helper2(df['gtotal'], df['annres'], 'ANN')
```