Report

Zid: z5235878

Name: Su Zhaokun

Course: comp9321

```
In [1]:
import pandas as pd
import numpy as np
import sys
import ast
import json
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsClassifier
from scipy.stats import pearsonr
from sklearn.metrics import *
from decimal import Decimal
```

Load Data

```
In [2]:
# $ python3 z{id}.py path1 path2
def load_data(path1, path2):
    df_train_origin = pd.read_csv(path1)
    df_validation_origin = pd.read_csv(path2)

    # shuffle all data
    df_train_origin = shuffle(df_train_origin)
    df_validation_origin = shuffle(df_validation_origin)
    return df_train_origin, df_validation_origin
```

Feature Engineering

Firstly, we need to make a relatively accurate prediction for movie "revenue" (part_1) and classify the rating for different movies (part_2) when it comes to our job.

Based on our basic assumption, in general, a high revenue means a good rating and vice versa. We cannot predict revenue using rateing and it is meaningless that make a classification for rateing using revenue.

By naturally thinking, I decide to use attributes list ["cast","crew","budget","original_language","genres","release_date","runtime"] to predict revenue since the remaining attributes are not very relevant to movie revenue or rating.

original dataset:

	cast	crew	budget	original_language	genres	release_date	runtime
48	[{"cast_id": 7, "character": "Jack", "credit_i...	[{"credit_id": "569396fb925141655700200d", "de...	195000000	en	[{"id": 28, "name": "Action"}, {"id": 10751, "...	2013-02-27	114.0
249	[{"cast_id": 6, "character": "Lemuel Gulliver"...	[{"credit_id": "584f66ab9251416f10002c8d", "de...	112000000	en	[{"id": 35, "name": "Comedy"}]	2010-12-25	85.0
1402	[{"cast_id": 10, "character": "Amelia \"Mia\" ...	[{"credit_id": "55783958c3a36842ee00177f", "de...	37000000	en	[{"id": 35, "name": "Comedy"}, {"id": 10751, "...	2001-08-03	115.0
2061	[{"cast_id": 1, "character": "Ryan Dunne", "cr...	[{"credit_id": "52fe450ec3a368484e045d67", "de...	34000000	en	[{"id": 18, "name": "Drama"}, {"id": 35, "name...	2001-08-22	108.0
667	[{"cast_id": 1, "character": "Paul Brenner", "...	[{"credit_id": "56904a209251414570000083", "de...	60000000	en	[{"id": 80, "name": "Crime"}, {"id": 18, "name...	1999-06-18	116.0
111	[{"cast_id": 42, "character": "Alexander", "cr...	[{"credit_id": "53943d0dc3a3684252000c2c", "de...	155000000	en	[{"id": 10752, "name": "War"}, {"id": 36, "nam...	2004-11-21	175.0
858	[{"cast_id": 6, "character": "Schmidt", "credi...	[{"credit_id": "5635fa64925141284c01a325", "de...	50000000	en	[{"id": 80, "name": "Crime"}, {"id": 35, "name...	2014-06-05	112.0
1409	[{"cast_id": 1, "character": "Paul Vitti", "cr...	[{"credit_id": "52fe4506c3a36847f80b7c81", "de...	80000000	en	[{"id": 35, "name": "Comedy"}, {"id": 80, "nam...	1999-03-05	103.0
1775	[{"cast_id": 1, "character": "Jimmy Morris", "...	[{"credit_id": "52fe460a9251416c7506b15b", "de...	20000000	en	[{"id": 18, "name": "Drama"}, {"id": 10751, "n...	2002-03-25	127.0
1426	[{"cast_id": 40, "character": "Dan Mott", "cre...	[{"credit_id": "52fe43b29251416c7501a909", "de...	19000000	en	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	2004-08-20	95.0

after shuffling (we made a random permutation for our original dataset):

	cast	crew	budget	original_language	genres	release_date	runtime
48	[{"cast_id": 7, "character": "Jack", "credit_i...	[{"credit_id": "569396fb925141655700200d", "de...	195000000	en	[{"id": 28, "name": "Action"}, {"id": 10751, "...	2013-02-27	114.0
249	[{"cast_id": 6, "character": "Lemuel Gulliver"...	[{"credit_id": "584f66ab9251416f10002c8d", "de...	112000000	en	[{"id": 35, "name": "Comedy"}]	2010-12-25	85.0
1402	[{"cast_id": 10, "character": "Amelia \"Mia\" ...	[{"credit_id": "55783958c3a36842ee00177f", "de...	37000000	en	[{"id": 35, "name": "Comedy"}, {"id": 10751, "...	2001-08-03	115.0
2061	[{"cast_id": 1, "character": "Ryan Dunne", "cr...	[{"credit_id": "52fe450ec3a368484e045d67", "de...	34000000	en	[{"id": 18, "name": "Drama"}, {"id": 35, "name...	2001-08-22	108.0
667	[{"cast_id": 1, "character": "Paul Brenner", "...	[{"credit_id": "56904a209251414570000083", "de...	60000000	en	[{"id": 80, "name": "Crime"}, {"id": 18, "name...	1999-06-18	116.0
111	[{"cast_id": 42, "character": "Alexander", "cr...	[{"credit_id": "53943d0dc3a3684252000c2c", "de...	155000000	en	[{"id": 10752, "name": "War"}, {"id": 36, "nam...	2004-11-21	175.0
858	[{"cast_id": 6, "character": "Schmidt", "credi...	[{"credit_id": "5635fa64925141284c01a325", "de...	50000000	en	[{"id": 80, "name": "Crime"}, {"id": 35, "name...	2014-06-05	112.0
1409	[{"cast_id": 1, "character": "Paul Vitti", "cr...	[{"credit_id": "52fe4506c3a36847f80b7c81", "de...	80000000	en	[{"id": 35, "name": "Comedy"}, {"id": 80, "nam...	1999-03-05	103.0
1775	[{"cast_id": 1, "character": "Jimmy Morris", "...	[{"credit_id": "52fe460a9251416c7506b15b", "de...	20000000	en	[{"id": 18, "name": "Drama"}, {"id": 10751, "n...	2002-03-25	127.0
1426	[{"cast_id": 40, "character": "Dan Mott", "cre...	[{"credit_id": "52fe43b29251416c7501a909", "de...	19000000	en	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	2004-08-20	95.0

```
In [3]:
def feature_engineering(df_train_origin, df_validation_origin):
    feature_columns = ["cast", "crew", "budget", "original_language", "genres", "release_date", "runtime"]
    df_train_X = df_train_origin[feature_columns]  # part_1 training data X
    df_validation_X = df_validation_origin[feature_columns]  # part_1 validation data X
    return df_train_X, df_validation_X
```

Data Processing

Now, we get all attributes that we want to use to make regression and classification, the next thing is that we should convert data to what we can use to our model and change some format data

For "cast" column, I choose the name of the leader star as feature, and encoding it, since the leader actor's name is significantly relevant for its value.

For "crew" column, I choose the name of the Director as feature because Director normally influence revenue of a movie for ordinary people, and encoding it.

For "budget" column, I can directly use original data.

For "original_language" column, I encode different language with different code.

For "genres" column, I use the first type of a movie as our feature and encode it.

For "release_date" column, considering the specific year and date are nothing to do with revenue, so I extract the month value to train our model

For "runtime" column, I can directly use original data.

datasets after pre-processing:

	cast	crew	budget	original_language	genres	release_date	runtime
1184	Charlie Tahan	Tim Burton	39000000	en	16	10	87.0
1854	Jim Cummings	Jun Falkenstein	30000000	en	16	2	77.0
1563	Morgan Freeman	Gary Fleder	27000000	en	18	10	115.0
910	Sandra Bullock	Barbet Schroeder	50000000	en	80	4	120.0
1483	Josh Hartnett	Paul McGuigan	30000000	en	18	9	114.0
541	Bryce Dallas Howard	M. Night Shyamalan	60000000	en	18	7	108.0
1324	Elijah Wood	Richard Donner	35000000	en	18	2	114.0
1673	Mark Wahlberg	Rupert Wyatt	25000000	en	53	12	111.0
24	Naomi Watts	Peter Jackson	207000000	en	12	12	187.0
107	Arnold Schwarzenegger	Alan Taylor	155000000	en	878	6	126.0

In [4]:
```python
def data_pre_processing(df_train_X, df_validation_X):
    # regulazition all data sets

    # step 1: extract cast lead name in cast
    # for training data
    lead_star_list_train = []  # a list stores all leading stars
    for cast in df_train_X['cast']:
        cast = ast.literal_eval(cast)
        lead_star_list_train.append(cast[0]['name'])
    df_train_X['cast'] = lead_star_list_train

    # for testing data
    lead_star_list_validation = []  # a list stores all leading stars
    for cast in df_validation_X['cast']:
        cast = ast.literal_eval(cast)
        lead_star_list_validation.append(cast[0]['name'])
    df_validation_X['cast'] = lead_star_list_validation

    # step 2: extract director name in crew
    # for training data
    director_list_train = []  # a list stores all leading stars
    for crew in df_train_X['crew']:
        crew = ast.literal_eval(crew)
        for member in crew:
            if member['job'] == "Director":
                director_list_train.append(member['name'])
                break
    df_train_X['crew'] = director_list_train

    # for testing data
    director_list_validation = []  # a list stores all leading stars
    for crew in df_validation_X['crew']:
        crew = ast.literal_eval(crew)
        for member in crew:
            if member['job'] == "Director":
                director_list_validation.append(member['name'])
                break
    df_validation_X['crew'] = director_list_validation

    # step 3: extract main genres
    # for training data
    genres_list_train = []
    for pc in df_train_X['genres']:
        pc = ast.literal_eval(pc)
        genres_list_train.append(pc[0]['id'])
    df_train_X['genres'] = genres_list_train
    # for testing data
    genres_list_validation = []
    for pc in df_validation_X['genres']:
        pc = ast.literal_eval(pc)
        genres_list_validation.append(pc[0]['id'])
    df_validation_X['genres'] = genres_list_validation

    # step 4: extract all months
    # for training data
    month_train = []
    for date in df_train_X['release_date']:
        month_train.append(int(date[5:7]))
    df_train_X['release_date'] = month_train

    # for testing data
    month_validation = []
    for date in df_validation_X['release_date']:
        month_validation.append(int(date[5:7]))
    df_validation_X['release_date'] = month_validation

    return df_train_X, df_validation_X
```

Format Converting (Encoding)

After pre-processing our data sets, the next thing needs to handle is encoding.

For columns ["cast", "crew", "original_language", "genres"], we need to convert all those names and language labels into numbers. we encode those data by using integers.(e.g. we can encode 13 different kinds of language labels with integer from 1 to 13), Similiarly, we can encode other attributes using the same way.

dataset after encoding:

	cast	crew	budget	original_language	genres	release_date	runtime
1184	142	995	39000000	4	16	10	87.0
1854	422	539	30000000	4	16	2	77.0
1563	638	308	27000000	4	18	10	115.0
910	769	67	50000000	4	80	4	120.0
1483	464	746	30000000	4	18	9	114.0
541	120	606	60000000	4	18	7	108.0
1324	257	808	35000000	4	18	2	114.0
1673	579	871	25000000	4	53	12	111.0
24	639	766	207000000	4	12	12	187.0
107	63	13	155000000	4	878	6	126.0

In [5]:
```python
def data_encoding(df_train_X, df_validation_X):
    # encoding data
    cast_set = set(df_train_X['cast']).union(set(df_validation_X['cast']))
    cast_dict = dict()
    cast_list = sorted(list(cast_set))
    index = 1
    for name in cast_list:
        cast_dict[name] = index
        index += 1
    # convert name to id
    for i in range(len(df_train_X)):
        name = df_train_X.loc[i, 'cast']
        df_train_X.loc[i, 'cast'] = cast_dict[name]
    for i in range(len(df_validation_X)):
        name = df_validation_X.loc[i, 'cast']
        df_validation_X.loc[i, 'cast'] = cast_dict[name]

    crew_set = set(df_train_X['crew']).union(set(df_validation_X['crew']))
    crew_dict = dict()
    crew_list = sorted(list(crew_set))
    index = 1
    for name in crew_list:
        crew_dict[name] = index
        index += 1
    # convert name to id
    for i in range(len(df_train_X)):
        name = df_train_X.loc[i, 'crew']
        df_train_X.loc[i, 'crew'] = crew_dict[name]
    for i in range(len(df_validation_X)):
        name = df_validation_X.loc[i, 'crew']
        df_validation_X.loc[i, 'crew'] = crew_dict[name]

    language_set = set(df_train_X['original_language']).union(set(df_validation_X['original_language']))
    language_dict = dict()
    language_list = sorted(list(language_set))
    index = 1
    for name in language_list:
        language_dict[name] = index
        index += 1

    # convert name to id
    for i in range(len(df_train_X)):
        name = df_train_X.loc[i, 'original_language']
        df_train_X.loc[i, 'original_language'] = language_dict[name]
    for i in range(len(df_validation_X)):
        name = df_validation_X.loc[i, 'original_language']
        df_validation_X.loc[i, 'original_language'] = language_dict[name]
    return df_train_X, df_validation_X
```

Train Our Model And Show Results

For part1, we use LinearRegression Model to train our data sets and predict value of revenue of a specific movie.

For part2, we use K Nearest Neighbors Model to train our data sets and predict the rank of a specific movie.

After that, we get all results of our model for part_1:

MSE: 9660489995052654.0

correlation: 0.1 (around)

After that, we get all results of our model for part_2:

precision_score: 0.84 (around)

recall_score: 0.70 (around)

accuracy_score: 0.70 (around)

In [6]:
```
df_train_origin, df_validation_origin = load_data("training.csv", "validation.csv")
df_train_X, df_validation_X = feature_engineering(df_train_origin, df_validation_origin)
df_train_X, df_validation_X = data_pre_processing(df_train_X, df_validation_X)
df_train_X, df_validation_X = data_encoding(df_train_X, df_validation_X)

# part1 model
model1 = LinearRegression()
model1.fit(df_train_X, df_train_origin['revenue'])
predicted_y_part1 = model1.predict(df_validation_X)

# generate summary.csv for part1
MSR = mean_squared_error(df_validation_origin['revenue'], predicted_y_part1)
correlation = round(pearsonr(df_validation_origin['revenue'], predicted_y_part1)[0], 2)  # a tuple (correlaction, R-value)
```

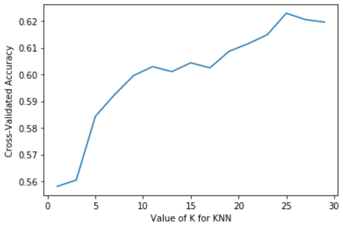
Using cross validation to train our model:

generate a figure below to support our choice

In [7]:
```
# part2 model
k_range = range(1, 31, 2)
k_scores = []

# iterate different k to determine the best k with the best perfromance
# using cross validation
for k in k_range:
    model2 = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(model2, df_train_X, df_train_origin['rating'], cv=10, scoring='accuracy')
    k_scores.append(scores.mean())

# plot
plt.plot(k_range, k_scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Cross-Validated Accuracy')
plt.show()
```



In [8]:
```
best_k = k_range[k_scores.index(max(k_scores))]
print("best k we use for KNN Classifier is: ", best_k)
final_k = 19
model2 = KNeighborsClassifier(n_neighbors=final_k)
model2.fit(df_train_X, df_train_origin['rating'])
predicted_y_part2 = model2.predict(df_validation_X)

# generate summary.csv for part2
# Decimal(a).quantize(Decimal("0.00"))
average_precision = Decimal(round(precision_score(df_validation_origin['rating'], predicted_y_part2, average="macro"), 3)).qu
average_recall = Decimal(round(recall_score(df_validation_origin['rating'], predicted_y_part2, average="weighted"), 2)).quant
accuracy = Decimal(round(accuracy_score(df_validation_origin['rating'], predicted_y_part2), 2)).quantize(Decimal("0.00"))
```

```
best k we use for KNN Classifier is:  25
```

In [9]:
```
print("MSE: ", MSR)
print("correlation: ", correlation)
```

```
MSE:  96604899950570002.0
correlation:  0.09
```

In [10]:
```
# print(classification_report(df_validation_origin['rating'], predicted_y_part2))
print("precision_score: ", average_precision)
print("recall_score: ", average_recall)
print("accuracy_score: ", accuracy)
```

```
precision_score:  0.85
recall_score:  0.70
accuracy_score:  0.70
```

In [ ]: