Milestone 4 Finalizing Results

February 12, 2023

1 Predictive Analytics: Milestone 4

Joshua Greenert, Gabriel Avinaz, and Mithil Patel

DSC630-T301 Predictive Analytics

2/10/2023

1.0.1 Milestone 2: Data Selection and Proposal

Introduction Since antiquity, games have been an integral aspect of human society – especially for cultural development and social interaction. With the advancement of human civilization, the way games are being played has drastically changed over the eons as modern games are primarily played electronically (video games). With over 3 billion players worldwide and approximately 200 billion dollars in revenue, the video gaming industry is constantly looking to attract new customers and boost playtime. As a result, we will implement a collaborative recommendation system to assist users in finding similar games that may interest them. By utilizing a collaborative filtering system, we can use historical review information to determine whether a product might interest one of our customers. We will experiment with two approaches to determine which application would be better for an e-commerce implementation.

What types of model(s) do you plan to use and why, and how do you intend to evaluate your results? We will implement a collaborative recommendation system to assist users in finding similar games that may interest them. By utilizing a collaborative filtering system, we can use historical review information to determine whether a product might interest one of our customers. We will experiment with two approaches to determine which application would be better for an e-commerce implementation.

Our first approach will implement memory-based collaborative filtering, which utilizes the collective review information of other users to find similar games. We'll be implementing both item-item and user-item methods to determine which provides a better result for our users. These two methods offer predictions based on what similar users like and what users who like a particular game are likely to enjoy. We'll use several distance measurement algorithms to determine which provides a more accurate recommendation such as: cosine similarity, Pearson Correlation, and K-nearest neighbors. The models created using these methods should perform fastest while providing acceptable accuracy. For our model-based collaborative filtering, we will train a few machine learning (ML) algorithms to make recommendations and find similar games. Our project will focus on utilizing matrix factorization algorithms to make this determination but will be looking to implement a deep learning approach given the time. The process of applying different ML algorithms is fairly simple, so we'll be looking to implement the most we can: SVD, PMF, and NMF, then compare our results with the memory-based results.

Our second approach to creating a recommendation system for our games library is by implementing a content-based filtering system utilizing natural language processing and available user reviews. This process will use a vectorized matrix of user reviews to find games that have been reviewed in a similar way. We can implement many of the techniques from our memory-based collaborative filtering process into this one and measure performance across each of our distance-measuring algorithms. We can utilize a standard train-test methodology from many of the of the models we will be implementing while cross referencing our suggested games with user-rated games. We also have user information in our data set as to whether they would make a recommendation for the game they are reviewing. We can utilize this data in testing our results accuracy.

What do you hope to learn? We hope to learn consumers' interests and buying patterns which can help our company optimize revenue by focusing resources on developing games customers will more likely enjoy. Additionally, we intend to learn appropriate methods to develop recommendation systems for alternative enterprises and future outsourcing. Finally, while going through this process, we expect to learn data from our models that can be used for other points of interest for the company and our personal career development.

Are there any risks or ethical implications with your proposal? Fortunately, all of the personal information of each individual review has been stripped from the data provided from the Steam API. Instead of the actual user's profile information, a review_id value is used instead. However, the comments may contain personal or sensitive information which will be appropriately handled or removed to ensure the safety and security of all users. Besides potential user-provided information, there are no additional risks or ethical concerns within this project.

What is your contingency plan? In the event that our findings aren't conclusive, our dataset proves too massive to work with, or our data doesn't express a proper recommendation system as intended, we have collected another dataset to use instead. This database revolves around used cars posted to Craigslist and would have a similar strategy to our current proposal. We would use the dataset to review models for recommendations for users based upon their interests and preferences. From that information, we would predict cars that users would be interested in while also being local to their respective area based on the information from the dataset. All parties have agreed that if our project doesn't appear feasible by the end of week 3, we will be shifting gears to this alternative to salvage the time we have remaining.

Additional important information As the project progresses, we plan to implement a deep learning algorithm if we are able to add it into the scope; at the moment, we anticipate our limited time will be a factor that prevents us from doing so. With a deep learning algorithm, we would be able to enhance our accuracy and predict better recommendations for our users.

1.0.2 Milestone 3: Preliminary Analysis

Will I be able to answer the questions I want to answer with the data I have? The data are proving to be far more time-consuming than initially expected. Nevertheless, the data should be more than capable of providing a model of a working recommendation system that can suggest games to users based on their game preference. Additionally, our team has utilized a supplemental dataset to create alternative options for our recommendations.

What visualizations are especially useful for explaining my data? Histogram plots were one of the most useful visualizations in evaluating our data. It helped us determine that a large number of users that left reviews on the steam application would only leave a review for one game. This has a significant effect on how we can treat the data going forward, especially with the collaborative filtering models we were going to attempt to implement. Bar plots can be used to display our model's outputs, showing ratios of similarity with each game. Scatterplots were one of the more important visualizations in determining how to measure our aggregate review data for each game. It helped us determine how the average score changed between positive reviews and those weighted for quality.

Do I need to adjust the data and/or driving questions? Due to the size of our data selected, we have implemented a checkpoint inside of our file where a csv file is exported after a rigorous six-hour lemmatization. This export can then be used to continue the process of creating the remainder of the model while utilizing the results of the previous steps. While an adjustment to the process and data was a serious consideration, the checkpoint simplifies the approach and allows us to begin further development without the need to start from the beginning.

Do I need to adjust my model/evaluation choices? For the most part, our models and methodology appear to be functioning as intended. Our intention is to use Term Frequency-Inverse Document Frequency (TFIDF) vectorized user reviews and game descriptions with cosine similarity as a metric to measure the closeness of each recommendation. Additionally, we plan to have a collaborative filtering model utilizing K Nearest Neighbor (KNN), random forest, and XGBoost as classifiers based on the user's review history. The expectation is to utilize GridSearchCV to find the ideal model and parameters for this method. Nothing at the moment appears to require a significant change, but we learned from our visualizations that we can gain a significant amount of efficiency by dropping users that have only contributed a single review from the collaborative dataset, and we will still have plenty of entries after the fact to split up data into training and test sets.

Are my original expectations still reasonable? Our initial expectations were to create a prediction model that could make recommendations based on the reviews from Steam users. This expectation still shows promise as we've completed the majority of the modeling process. However, the expectation to create a neural network with the dataset may prove challenging based on the size of the dataset, and how long the process took for lemmatization. Alternatively, we've chosen to include an alternative dataset for an additional option on recommendation systems.

1.0.3 Milestone 4: Finalizing Results

Explain your process for preparing the data Our initial dataset contained a massive amount of rows and comments from Steam users. However, most of the reviews were in other languages. First, we removed all of the other languages besides "english" to ensure we could use and understand the results of the model. Next, we dropped columns that we estimated were insignificant to our process and made dummies for the remaining categorical variables. We imported libraries for keyword sentiment analysis and proceeded to lemmatize the comments from each user to remove stop words and contractions; this process (in code) took over 6 hours! Afterwards, we created a checkpoint to begin the final processes with our dataset by adding another dataset to use for cosine similarity. Here are some of the steps performed in code:

1.0.4 DO NOT RUN: ONLY FOR VISUAL PURPOSES!

[ ]:
```
%%script false --no-raise-error
# Using the language column, we can remove all other languages besides english.
df_steam_reviews = df_steam_reviews[df_steam_reviews['language'] == 'english']

# Drop the columns that we don't need.
df_steam_reviews = df_steam_reviews.drop(['Unnamed: 0', 'review_id',␣
 ↪'language','author.num_games_owned',
                                          'author.last_played' ], axis = 1)

# Make dummies of the columns that can conform.
df_reviews_dummies = pd.get_dummies(df_steam_reviews, columns=['recommended',␣
 ↪'steam_purchase', 'received_for_free',
                                                                          ␣
 ↪'written_during_early_access'])

# Calling the process_sentence_lemmatize function through the apply method;␣
 ↪skipped some steps to reduce file size.
# This is the part that took 6 hours!
df_reviews_dummies['prepped_review_lemm'] = df_reviews_dummies.review.
 ↪apply(process_sentence_lemmatize)

# Calling the process_sentence_lemmatize function through the apply method.
df_steam_games['prepped_description_lemm'] = df_steam_games['Short␣
 ↪Description'].apply(process_sentence_lemmatize)
```

```
Couldn't find program: 'false'
```

Build and evaluate at least one model

[ ]:
```
# Import the required libaries.
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.neighbors import NearestNeighbors
from scipy.sparse import csr_matrix
```

[ ]:
```
# Pull in the dataframe.

# Josh G
# df_reviews_dummies = pd.read_csv('../../../../../Downloads/Prepped_test_out.
 ↪csv', on_bad_lines="skip", engine="python")
```

```
# df_reviews = pd.read_csv('../../../../../Downloads/steam_reviews.csv',␣
 ↪low_memory=False)
# df_steam_games = pd.read_csv('../../../../../Downloads/Prepped_games_out.csv')
# df_steam_games = df_steam_games.set_index('App ID')

# Gabe
df_reviews_dummies = pd.read_csv('Prepped_test_out.csv')
df_reviews = pd.read_csv('steam_reviews.csv')
df_steam_games = pd.read_csv('Prepped_games_out.csv')
df_steam_games = df_steam_games.set_index('App ID')
```

As noted above, we brought in some new datasets to use for comparison on our models generated. The goal was to create a model that was fairly consistent to compare against the lemmatized model we would use later on.

```
[ ]: # merge the two columns into one
df_reviews_dummies["recommended"] = df_reviews_dummies['recommended_False'] +␣
 ↪df_reviews_dummies['recommended_True']

# Set the user recommendation dataframes.
user_rec_df_reviews = df_reviews[["app_id","author.steamid", "recommended"]].
 ↪copy()
user_rec_df_prepped = df_reviews_dummies[["app_id","author.steamid",␣
 ↪"recommended"]].copy()
```

```
[ ]: # Set the values that will be used for the matrix dimensions.
n_ratings_reviews = len(user_rec_df_reviews)
n_games_reviews = len(user_rec_df_reviews['app_id'].unique())
n_users_reviews = len(user_rec_df_reviews['author.steamid'].unique())

n_ratings_prepped = len(user_rec_df_prepped)
n_games_prepped = len(user_rec_df_prepped['app_id'].unique())
n_users_prepped = len(user_rec_df_prepped['author.steamid'].unique())

# Print the results to confirm they have values.
print(n_ratings_reviews, n_games_reviews, n_users_reviews)
print(n_ratings_prepped, n_games_prepped, n_users_prepped)
```

```
21747371 315 12406560
9635437 315 5287718
```

```
[ ]: # maps idices to users and game IDs
user_map_reviews = dict(zip(np.unique(user_rec_df_reviews["author.steamid"]),␣
 ↪list(range(n_users_reviews))))
game_map_reviews = dict(zip(np.unique(user_rec_df_reviews["app_id"]),␣
 ↪list(range(n_games_reviews))))
```

```
# Confirm that the total reviews equal the n_games_reviews. Should be true,␣
 ↪true.
print(len(user_map_reviews) == n_users_reviews)
print(len(game_map_reviews) == n_games_reviews)

user_i_map_reviews = dict(zip(list(range(n_users_reviews)), np.
 ↪unique(user_rec_df_reviews["author.steamid"])))
game_i_map_reviews = dict(zip(list(range(n_games_reviews)), np.
 ↪unique(user_rec_df_reviews["app_id"])))

# Perform the same operations on the prepped items.
user_map_prepped = dict(zip(np.unique(user_rec_df_prepped["author.steamid"]),␣
 ↪list(range(n_users_prepped))))
game_map_prepped = dict(zip(np.unique(user_rec_df_prepped["app_id"]),␣
 ↪list(range(n_games_prepped))))

# Confirm that the total reviews equal the n_games_reviews. Should be true,␣
 ↪true.
print(len(user_map_prepped) == n_users_prepped)
print(len(game_map_prepped) == n_games_prepped)

user_i_map_prepped = dict(zip(list(range(n_users_prepped)), np.
 ↪unique(user_rec_df_prepped["author.steamid"])))
game_i_map_prepped = dict(zip(list(range(n_games_prepped)), np.
 ↪unique(user_rec_df_prepped["app_id"])))
```

```
True
True
True
True
```

```
[ ]: # creating indices for csr_matrix
user_index_reviews = [user_map_reviews[i] for i in user_rec_df_reviews['author.
 ↪steamid']]
game_index_reviews = [game_map_reviews[i] for i in␣
 ↪user_rec_df_reviews['app_id']]

user_index_prepped = user_rec_df_prepped['author.steamid'].
 ↪map(user_map_prepped, na_action='ignore').fillna(-1).astype(int)
game_index_prepped = [game_map_prepped[i] for i in␣
 ↪user_rec_df_prepped['app_id']]
```

```
[ ]: # creates csr matrixes
matrix_reviews = csr_matrix((user_rec_df_reviews["recommended"],␣
 ↪(game_index_reviews, user_index_reviews)), shape=(n_games_reviews,␣
 ↪n_users_reviews))
```

```
matrix_prepped = csr_matrix((user_rec_df_prepped["recommended"],␣
 ↪(game_index_prepped, user_index_prepped)), shape=(n_games_prepped,␣
 ↪n_users_prepped))
```

[ ]:
```
# creating a dictionary to pull the name of games from based on IDs
game_names_reviews = dict(zip(df_reviews['app_id'], df_reviews['app_name']))
game_names_prepped = dict(zip(df_reviews_dummies['app_id'],␣
 ↪df_reviews_dummies['app_name']))
```

[ ]:
```
'''
This function uses both datasets to determine the best match for returning a␣
 ↪list of games.  It uses KNN
along with the cosine function to find similiaries and store them into a list␣
 ↪of neighbor ids.  These ids
are then compared for accuracy and the highest matches are returned.

@param game_id: the id that references the game found with Steam.
@param total_matches: the total number of matches to return
@param usePrepped: boolean value to determine if usePrepped should be used in␣
 ↪this test model.
'''
def find_related_games(game_id, total_matches, usePrepped):

    # Set variables
    reviews_neighbour_ids_with_distance = {}

    # Prepare index/vectorizations.
    game_index_reviews = game_map_reviews[game_id]
    game_vector_reviews = matrix_reviews[game_index_reviews]

    # Increment total matches.
    total_matches += 1

    # Set the KNN model and fit them.
    kNN_reviews = NearestNeighbors(algorithm = 'brute', metric='cosine')
    kNN_reviews.fit(matrix_reviews)

    # reshape and determine distances for KNN values.
    game_vec_reshaped_reviews = game_vector_reviews.reshape(1,-1)
    distances_reviews, indices_reviews = kNN_reviews.
 ↪kneighbors(game_vec_reshaped_reviews, n_neighbors=total_matches)

    # Loop through and flatten the distances provided
    for i in range(0,len(distances_reviews.flatten())):
        n = indices_reviews.flatten()[i]
        neighbour_id = game_i_map_reviews[n]
        reviews_neighbour_ids_with_distance[game_names_reviews[neighbour_id]] =␣
↪distances_reviews.flatten()[i]
    reviews_neighbour_ids_with_distance.pop(game_names_reviews[game_id], None)␣
↪# removes the same game

    # Sort the data by accuracy
    sorted_neighbour_ids_with_distance_reviews =␣
↪sorted(reviews_neighbour_ids_with_distance.items(), key=lambda x: x[1],␣
↪reverse=True)
    if usePrepped:
        sorted_neighbour_ids_with_distance_prepped =␣
↪find_related_games_usePrepped(game_id, total_matches)
    else:
        sorted_neighbour_ids_with_distance_prepped = []

    # finalize the list amongst the two created.
    combined_list = sorted_neighbour_ids_with_distance_reviews +␣
↪sorted_neighbour_ids_with_distance_prepped
    sorted_combined_list = sorted(combined_list, key=lambda x: x[1])

    # Print the games and their related accuracy.
    count = 1

    print(f"Games related to: {game_names_reviews[game_id]}\n")
    for game_name, accuracy in sorted_combined_list:
        if count == total_matches:
            break
        else:
            print(f"{game_name}: {accuracy:.2f}")
            count += 1

'''
This method breaks the logic into a separate function to keep the prior␣
↪function clean of unnecessary if statements.
returns a sorted neighbor_ids list.
'''
def find_related_games_usePrepped(game_id, total_matches):
    prepped_neighbour_ids_with_distance = {}

    # Prepare index/vectorizations.
    game_index_prepped = game_map_prepped[game_id]
    game_vector_prepped = matrix_prepped[game_index_prepped]

    # Set the KNN model and fit them.
    kNN_prepped = NearestNeighbors(algorithm = 'brute', metric='cosine')
    kNN_prepped.fit(matrix_prepped)
    # reshape and determine distances for KNN values.
    game_vec_reshaped_prepped = game_vector_prepped.reshape(1,-1)
    distances_prepped, indices_prepped = kNN_prepped.
↪kneighbors(game_vec_reshaped_prepped, n_neighbors=total_matches)

    # Loop through and flatten the distances provided
    for i in range(0,len(distances_prepped.flatten())):
        n = indices_prepped.flatten()[i]
        neighbour_id = game_i_map_prepped[n]
        prepped_neighbour_ids_with_distance[game_names_prepped[neighbour_id]] =␣
↪distances_prepped.flatten()[i]
    prepped_neighbour_ids_with_distance.pop(game_names_prepped[game_id], None)

    # Sort the results.
    sorted_neighbour_ids_with_distance_prepped =␣
↪sorted(prepped_neighbour_ids_with_distance.items(), key=lambda x: x[1])

    return sorted_neighbour_ids_with_distance_prepped
```

```
[ ]: # Set a test game id and run the function.
     game_id = 292030
     total_matches = 5

     find_related_games(game_id, total_matches, False)
```

```
Games related to: The Witcher 3: Wild Hunt

DARK SOULS  III: 0.92
Rise of the Tomb Raider: 0.92
Dying Light: 0.93
Grand Theft Auto V: 0.93
Fallout 4: 0.93
```

```
[ ]: # Set a test game id and run the function.
     game_id = 292030
     total_matches = 5

     find_related_games(game_id, total_matches, True)
```

```
Games related to: The Witcher 3: Wild Hunt

Fallout 4: 0.92
DARK SOULS  III: 0.92
Rise of the Tomb Raider: 0.92
DOOM: 0.93
Dying Light: 0.93
```

The initial model above uses data from two separate datasets to return the lowest accuracies provided by each; a lower accuracy indicates less distance between each game for the KNN model. This method uses information that we can verify through our individual understanding of the games listed to reinforce whether our model is matching in a way that we think is acceptable (this step is a little more subjective since without playing the games listed, it can be difficult to determine if this is an accurate assessment). Whether the prepped data is true or false, the result set is matching. This indicates that both sets when coupled together will predict similarly and that we have a model can use for comparison.

For clarity, the game chosen, Witcher 3, is a 3D open world role-playing game where a user travels and completes various missions for money or resources. In addition to the main linear storyline, there are additional side quests to complete as well. In the five games returned above, each of the main details described for Witcher 3 are identical (3D, open world, free roam, etc.).

```
[ ]: # Create the td-idf vectorizer.
     tfidfvec = TfidfVectorizer()

     # Fit the model
     tfidf_gamedesc = tfidfvec.
      ↪fit_transform((df_steam_games['prepped_description_lemm'].values.
      ↪astype('U')))

     # Add cosine similarity
     cos_sim = cosine_similarity(tfidf_gamedesc, tfidf_gamedesc)

     # functions for retrieving similar game IDs from cos_sim
     indices = pd.Series(df_steam_games.index)
```

```
[ ]: '''
     Create a new function to get the data by the game's descriptions.
     Returns a list of recommended games from the model.

     @param: id: id of the game to get recommendations on.
     @param: total_matches: total number of matches to return.
     '''
     def get_related_games_by_desc(id, total_matches):

         # Set a new dataframe to return.
         recommended_games =  pd.DataFrame()

         # Set the index variable.
         index = indices[indices == id].index[0]

         # Increment total_matches to be 1 higher.
         total_matches += 1

         # Obtain the list of cosine similiarities and set them into a series.
         similarity_scores = pd.Series(cos_sim[index]).sort_values(ascending = False)
         top = list(similarity_scores.iloc[1:total_matches].index)
```

```
    recommended_games['Score'] = similarity_scores.iloc[1:total_matches]

    # Create a new list.
    app_ids = []

    # Loop over and set the top items in the list.
    for i in top:
        app_ids.append(list(df_steam_games.index)[i])
    recommended_games['App ID'] = app_ids

    print(f"Games related to: {df_steam_games.loc[df_steam_games.index ==␣
 ↪game_id].Name.values[0]}\n")
    for i, val in enumerate(recommended_games["App ID"]):
        print(f"{df_steam_games.loc[df_steam_games.index == val].Name.
 ↪values[0]}: {recommended_games.Score.values[i]:.2f}")
```

[ ]:
```
# Test the new tdidf model.
game_id = 221380
total_matches = 10

recommended_games = get_related_games_by_desc(game_id, total_matches)
```

```
Games related to: Age of Empires II (2013)

Age of Mythology: Extended Edition: 0.37
Rise of Nations: Extended Edition: 0.37
Spades: 0.26
Spectre: 0.23
Escape Legacy: Ancient Scrolls: 0.22
DRAKERZ-Confrontation: 0.21
Project Beril /     : 0.21
Age of Empires II: Definitive Edition: 0.21
Age of Wonders II: The Wizard's Throne: 0.21
Akin: 0.20
```

[ ]:
```
# Test the new tdidf model.
game_id = 10
total_matches = 10

recommended_games = get_related_games_by_desc(game_id, total_matches)
```

```
Games related to: Counter-Strike

Rise Of A Hero: 0.23
Operation swat: 0.23
BLACK CLOVER: QUARTET KNIGHTS: 0.22
Emily: Displaced: 0.22
Team Fortress Classic: 0.21
```

```
Rescue Party: Live!: 0.21
Werewolves Online: 0.21
Firefight!: 0.20
Audio Drive: 0.20
Infection: 0.20
```

```
[ ]: # Test the new tdidf model.
     game_id = 1271600
     total_matches = 10

     recommended_games = get_related_games_by_desc(game_id, total_matches)
```

```
Games related to: Panzer Crew VR

Steel Crew: 0.60
Tank Maniacs: 0.47
Tank Commander: Battlefield: 0.47
Steel Armor: Blaze of War: 0.46
Pocket Tank: 0.46
YUT YUT: 0.45
Tiger Tank 59 A-Gun: 0.44
Armored Battle Crew [World War 1] - Tank Warfare and Crew Management Simulator:
0.44
Panzer Panic VR: 0.44
Tank Force: 0.43
```

Interpret your results We decided that matrix factorization would be the best solution for both our content-based and collaborative filtering methods while finding a number of successful recommendations using both methods. Additionally, both methods utilized cosine distance as a measure of likeness to other games. For our collaborative model we used K-Nearest Neighbors (KNN) which utilizes cosine distances as a potential metric. The distance is measured as one minus the cosine similarity, so our results had to be inverted. The lower number resulted in more similar games while the higher numbers resulted in games with greater dissimilarity.

Our content-based recommendations utilized a TFIDFVectorized matrix of game descriptions to calculate it's cosine similarity. Unlike the KNN model, we did not have to reverse our list to accommodate for distance, and solely compared similarities with the highest similarities representing the most related games. Moreover, we did not use a separate dataset to compare our results with the TFIDF model. Regardless, both models provide results that can be useful for differing purposes.

Begin to formulate a conclusion/recommendations Our goal was to provide the best models for a number of different use cases, so we couldn't rely solely on one for each implementation. Our recommendation is that individual game pages can utilize our content-based solution — the TFIDFVectorized matrix — as this provides games that have very similar descriptions. From our results, we would often see expansions for games within our top results.

Alternatively, the collaborative solution — the KNN model — can work much more effectively as a recommendation of games on the user's homepage, often showing games that may or may not be related, but share a common purchase pattern across users.

As for recommendations given more time, we would recommend expansions for each type of model. Our collaborative filtering model could take several games and draw its similarities from them in creating personalized lists for users. Our content-based filtering model can expand it's matrices to include both genre and tag information in it's vectorized data.

1.0.5 Resources

Kaggle. (n.d.). All 55,000 Games on Steam (November 2022). Www.kaggle.com. Retrieved January 20, 2023, from https://www.kaggle.com/datasets/tristan581/all-55000-games-on-steam-november-2022

M., M. (2021) Steam reviews dataset 2021, Kaggle. Available at: https://www.kaggle.com/datasets/najzeko/steam-reviews-2021?select=steam_reviews.csv (Accessed: December 10, 2022)

Tristan. "All 55,000 Games on Steam (November 2022)." Www.kaggle.com, www.kaggle.com/datasets/tristan581/all-55000-games-on-steam-november-2022.