Report for the Term Project

CS 412 Machine Learning: Predicting HW grades from ChatGPT interactions.
19.01.2024

Overview

With this project, we aimed to predict the first assignment grades of the CS-412 course by using students' ChatGPT conversation histories.

Atahan Bozkuş - 28471
Emir Ay - 28150

Data Description

The data for our machine learning project consists of two main components:

- a collection of HTML files containing ChatGPT conversation histories (150-200 files)
- a score.csv file that correlates with the performance scores of the students.

The HTML files contain each student's detailed conversation histories, which serve as a historical record of their interactions with ChatGPT while completing assignments. Each HTML file is designed to capture the ongoing conversation, including the questions presented by the students and the responses provided by ChatGPT. These files effectively record the students' participation and interactions in the assignment-solving process. The score.csv file is an additional dataset including the equivalent scores obtained by each student in their homework assignments. This file contains the labeled data and provides a quantitative measure of the students' performance.

Code Description and Results

Cell 3: Includes several essential libraries and modules for the project.

Cell 4: Includes initiations of the NLTK downloads for stopwords and WordNet. The "preprocess_text function" performs a series of steps to clean and standardize text data.

- converts the text to lowercase,
- removes leading and trailing whitespaces,
- replaces non-alphanumeric characters with spaces,
- substitutes numbers with '<NUM>',
- tokenizes the text into words,
- removes common stop words,
- lemmatizes the remaining words.

This process ensures that the text data is in a consistent and refined format, which makes the data for feature extraction and machine learning model training.

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ataha\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\ataha\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Cell 5: The “extract_and_clean_code function” is designed to extract and clean code snippets from an HTML structure for further analysis or processing within the context of the HTML structure.

Cell 6: Reads HTML files containing ChatGPT conversation histories, extracting relevant information. This process enables the creation of a structured dataset mapping file codes to their respective conversations and code snippets for further analysis or model training.

```
100%|██████████| 127/127 [00:10<00:00, 11.62it/s]
```

Cell 7: The "clean_text" function is designed to clean and preprocess text data by removing non-ASCII characters and replacing multiple whitespaces with a single space. In the outer loop, it iterates over file codes and their respective conversations. The inner loop cleans the text content and code snippets using the clean_text function. This ensures that both the conversation text and code are appropriately preprocessed for further analysis or applications.

Cell 8: code2convos dictionary

```
code2convos
                                                                                                    Python
{'0031c86e-81f4-4eef-9e0e-28037abf9883': [{'role': 'user',
   'text': "Load a CSV file into a Pandas in Python. The file is named 'cs412_hw1_dataset.csv' and contains columns like 'Species', 'Island', 'Sex', 'Diet', 'Year', 'Life Stage', 'Body M
   'code': []},
  {'role': 'assistant',
   'text': "You can load a CSV file into a Pandas DataFrame in Python using the pandas.read_csv() function. Here's how you can do it for your specific CSV file:This code assumes that the
   'code': ["import pandas as pd file_path = 'cs412_hw1_dataset.csv' df = pd.read_csv(file_path) print(df.head())"]},
  {'role': 'user',
   'text': 'Provide Python code to understand a dataset using Pandas. Find the shape of the dataset, display variable names, display a summary of the dataset with the info() function, ar
   'code': []},
  {'role': 'assistant',
   'text': "You can use Pandas to understand a dataset by performing various exploratory data analysis (EDA) tasks. Here's a Python code snippet that accomplishes the tasks you mentioned
   'code': ['import pandas as pd file_path = \'cs412_hw1_dataset.csv\' df = pd.read_csv(file_path) shape = df.shape print("Shape of the dataset (rows, columns):", shape) variable_names =
  {'role': 'user',
   'text': "You will preprocess the data now 1. Check for missing values and handle them by either dropping or filling them with the most common values. Ensure that there is enough data
   'code': []},
  {'role': 'assistant',
   'text': "To preprocess the dataset as per your requirements, you can follow these steps:Check for missing values and handle them.Encode labels with mappings using the map function.Sp
   'code': ["import pandas as pd from sklearn.model_selection import train_test_split file_path = 'cs412_hw1_dataset.csv' df = pd.read_csv(file_path) most_common_sex = df['Sex'].mode()[(
  {'role': 'user',
   'text': "You already provide code for this but please recreate that part for shuffling if needed. 1. Shuffle dataframe named 'df'. 2. Separate the dataset into a feature matrix 'X' ar
   'code': []},
  {'role': 'assistant',
   'text': "Certainly, here's the code to shuffle the DataFrame, separate it into feature matrix 'X' and target vector 'y', and split the data into training and test sets with random shu
   'code': ["import pandas as pd from sklearn.model_selection import train_test_split file_path = 'cs412_hw1_dataset.csv' df = pd.read_csv(file_path) df = df.sample(frac=1, random_state=
  {'role': 'user',
...
   'code': []},
  {'role': 'assistant',
   'text': "The warning is indicating that there are still instances where the calculation encounters zero probabilities, resulting in NaN values. This could happen if the left or right
   'code': ['def calculate_entropy(probabilities): valid_probabilities = probabilities[probabilities != 0] if len(valid_probabilities) == 0: return 0 # If all probabilities are zero, ent
    'import numpy as np import pandas as pd from sklearn.tree import DecisionTreeClassifier, plot_tree from sklearn.model_selection import train_test_split from sklearn.model_selection :
```

Cell 9: Preprocessing of user prompts, which provides a clean and structured dataset for various natural language processing tasks.

- The "preprocess_text function" converts input text to lowercase and removes non-word characters.
- The "extract_user_prompts function" extracts user prompts from a list of conversations based on the role "user."

The code utilizes these functions to process and collect user prompts from the code2convos dictionary.

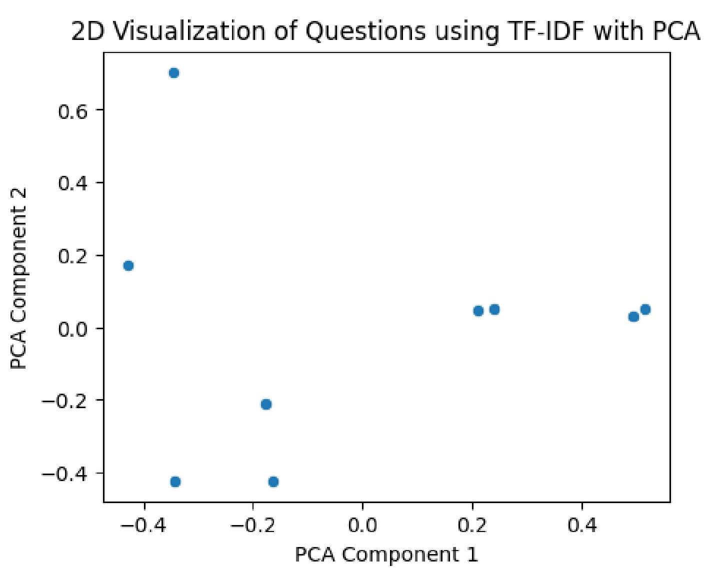
Cell 10:

```
print(prompts)
                                                                                   Python
["Load a CSV file into a Pandas in Python. The file is named 'cs412_hw1_dataset.csv' and contains columns like 'Species', 'Island', 'Sex', 'Diet', 'Year', 'I
```

Cell 11: Initialization of the task questions in Assignment.ipynb.

Cell 12: Utilizes the scikit-learn TfidfVectorizer to create a TF-IDF matrix from a combined collection of prompts and questions, with a specified text preprocessing function to standardize and clean the text data.

Cell 13: Transforms the vectorized representation of questions using TF-IDF into a 2D space through PCA and visualizes the result with a scatter plot using Seaborn.



Cell 14: Transforms the questions into a TF-IDF matrix, and converts the result into a Pandas DataFrame named questions_TF_IDF.

```python
questions_TF_IDF = pd.DataFrame(vectorizer.transform(questions).toarray(), columns=vectorizer.get_feature_names_out())
questions_TF_IDF.head()
```

Python

	00000	000000	000196	000282	000284	000360	000469	000750	000886	000991	...	youve	ypred	yticklabels	yticks	zaman	zdm	zero	zerodivisionerror
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 5361 columns

Cell 15: Processes user prompts associated with HTML file codes, computes their TF-IDF representations using TfidfVectorizer, and stores the results in a dictionary called code2prompts_TF_IDF. The aim is enabling a TF-IDF matrix representation for each code with user prompts for further analysis.

Cells 16 and 17:

```python
code2prompts_TF_IDF["089eb66d-4c3a-4f58-b98f-a3774a2efb34"].head()
```

Python

	00000	000000	000196	000282	000284	000360	000469	000750	000886	000991	...	youve	ypred	yticklabels	yticks	zaman	zdm	zero	zerodivisionerror
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 5361 columns

```python
code2prompts_TF_IDF["089eb66d-4c3a-4f58-b98f-a3774a2efb34"].shape
```

Python

```
(86, 5361)
```

Cell 18: Calculates the cosine similarity between the TF-IDF representations of questions and user prompts for each HTML file code, storing the results in a dictionary called code2cosine.

- An example putted on cell 19.

Cell 20: Mapping of the maximum cosine similarity scores of each question.

	code	Q_0	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8
0	0031c86e-81f4-4eef-9e0e-28037abf9883	0.153582	0.294878	0.522162	0.352245	0.460602	0.553879	0.167269	0.182931	0.195440
1	0225686d-b825-4cac-8691-3a3a5343df2b	0.192067	0.795607	0.772174	0.882656	0.607118	0.987511	0.892586	0.570733	0.544839
2	041f950b-c013-409a-a642-cffff60b9d4b	0.258378	0.295921	0.624849	0.351870	0.643034	0.454330	0.540284	0.546518	0.325791
3	04f91058-d0f8-4324-83b2-19c671f433dc	0.146047	0.117834	0.267381	0.316809	0.333889	0.314856	0.192434	0.261892	0.407217
4	089eb66d-4c3a-4f58-b98f-a3774a2efb34	0.344280	0.575528	0.782873	0.624833	0.724877	0.872171	0.684798	0.945305	0.512526
...	...	...	...	...	...	...	...	...	...	...
118	f24219d6-07f0-4baf-80ac-18475dc5b66f	0.187470	0.209888	0.358883	0.211246	0.149289	0.507131	1.000000	0.862585	0.739730
119	f2f18684-4a16-4c05-a2d1-c0f96d1de869	0.144801	0.139495	0.332274	0.950344	0.775707	0.828352	0.374145	0.905688	0.553340
120	f852596d-fdca-45aa-9050-d4f76ce6a53c	0.211060	0.908697	0.977726	0.924311	0.888609	0.921307	0.779558	0.767328	0.500262
121	f8ec3336-fd48-4654-ad98-62ccfb96d096	0.173262	0.967321	1.000000	0.703984	0.942662	0.907415	1.000000	1.000000	0.783744
122	fb8de815-224c-4d06-9fd4-7156d1a9920d	0.170342	0.712409	0.835224	0.760650	0.895230	0.943543	0.944887	0.844004	0.730284

Cell 21: Computes various features based on user prompts and ChatGPT responses and handles cases where there are no conversations.

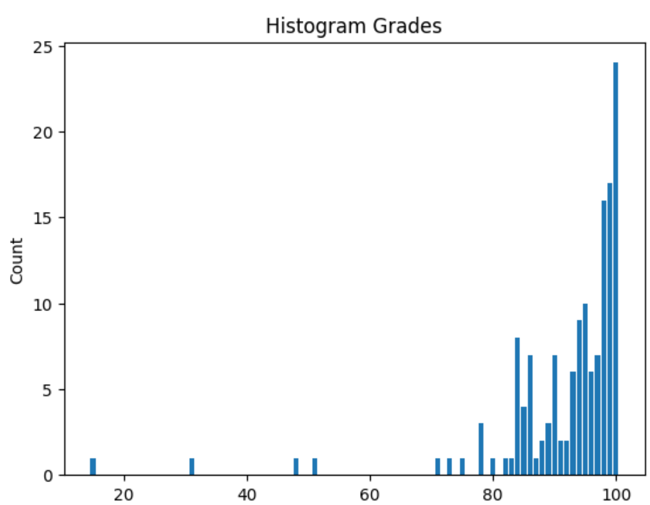
Cell 22: Transforms the dictionary code2features into a Pandas DataFrame.

	#user_prompts	#error	#no	#thank	#next	#entropy	#hata	#sorun	#anlamıyorum	prompt_avg_chars	response_avg_chars
0031c86e-81f4-4eef-9e0e-28037abf9883	14.0	3.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	405.071429	1061.142857
0225686d-b825-4cac-8691-3a3a5343df2b	18.0	0.0	0.0	0.0	0.0	3.0	0.0	0.0	0.0	252.000000	992.555556
041f950b-c013-409a-a642-cffff60b9d4b	9.0	3.0	0.0	1.0	0.0	3.0	0.0	0.0	0.0	644.777778	613.444444
04f91058-d0f8-4324-83b2-19c671f433dc	20.0	1.0	1.0	0.0	0.0	3.0	0.0	0.0	0.0	107.050000	962.550000

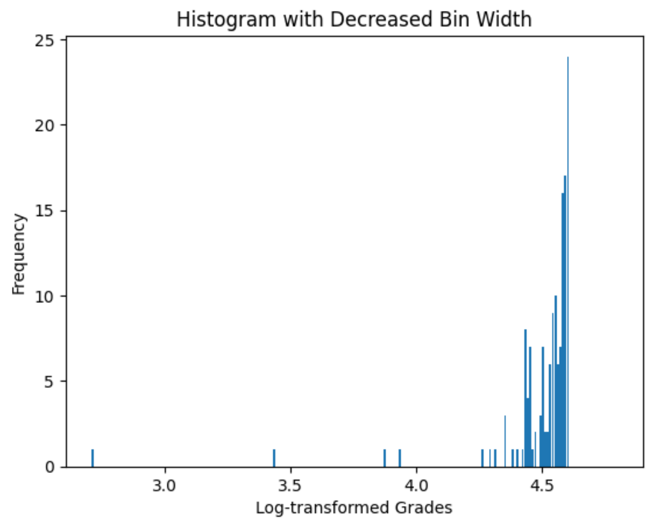
Cell 23: Reads the "scores.csv" into a Pandas DataFrame.

	code	grade
0	6a2003ad-a05a-41c9-9d48-e98491a90499	90.0
1	04f91058-d0f8-4324-83b2-19c671f433dc	97.0
2	81fdeb2a-e7e5-4a05-8058-d31ea579b0d9	94.0
3	6a903495-c5be-4263-b4dd-75e2bbc30434	97.0
4	6d5742c1-77c4-429c-8f6e-ef1262ca5557	93.0
5	dd898b12-b04b-404e-9b59-f846a162c177	98.0
6	dd898b12-b04b-404e-9b59-f846a162c177	98.0
7	b0640e51-6879-40cb-a4f5-329f952ef99d	98.0
8	668ad17e-0240-49f7-b5a7-d22e502554c6	99.0
9	ba18e4e8-2c26-46d4-ba31-cc21947aabd5	100.0

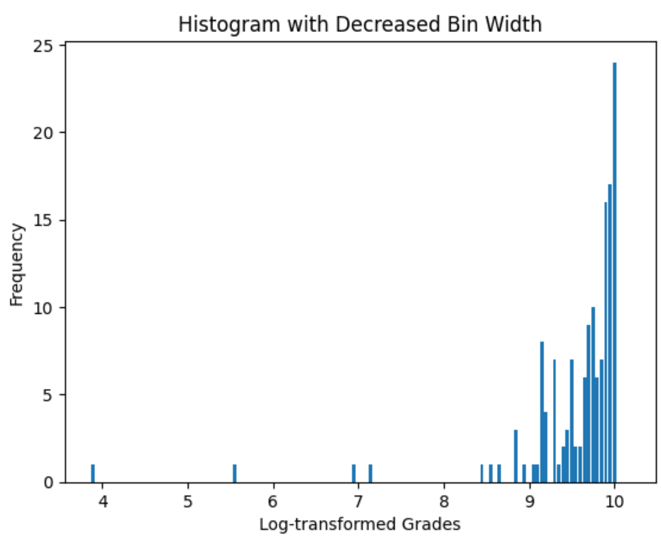
Cell 24: Generates a histogram to visualize the distribution of grades in the 'grade' column of the 'scores' DataFrame.



Cell 25: Performs a log transformation to the 'grade' column in the 'scores' DataFrame and creates a histogram with narrower bin widths to visually represent the distribution of log-transformed grades.



Cell 26: Performs a square root transformation to the 'grade' column in the 'scores' DataFrame and creates a histogram with narrower bin widths to visualize the distribution of square root-transformed grades.



Cell 27: Resets the index of the DataFrame and then renames the 'index' column to "code."

Cell 28: Merges the DataFrame 'df' with the 'question_mapping_scores' DataFrame based on the "code" column, using a left join. The resulting DataFrame now includes additional columns containing scores related to the similarity between questions and user prompts.

	code	#user_prompts	#error	#no	#thank	#next	#entropy	#hata	#sorun	#anlamıyorum	...	response_avg_chars	Q_0	Q_1	Q_2	Q_3
0	0031c86e-81f4-4eef-9e0e-28037abf9883	14.0	3.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	1061.142857	0.153582	0.294878	0.522162	0.352245
1	0225686d-b825-4cac-8691-3a3a5343df2b	18.0	0.0	0.0	0.0	0.0	3.0	0.0	0.0	0.0	...	992.555556	0.192067	0.795607	0.772174	0.882656
2	041f950b-c013-409a-a642-cffff60b9d4b	9.0	3.0	0.0	1.0	0.0	3.0	0.0	0.0	0.0	...	613.444444	0.258378	0.295921	0.624849	0.351870
3	04f91058-d0f8-4324-83b2-19c671f433dc	20.0	1.0	1.0	0.0	0.0	3.0	0.0	0.0	0.0	...	962.550000	0.146047	0.117834	0.267381	0.316809
4	089eb66d-4c3a-4f58-b98f-a3774a2efb34	86.0	1.0	5.0	0.0	0.0	26.0	0.0	0.0	0.0	...	762.883721	0.344280	0.575528	0.782873	0.624833

Cell 29: The dropna() function is applied to remove any rows with missing values, and the drop_duplicates method is used to keep only the first occurrence of each unique "code.

Python

	code	#user_prompts	#error	#no	#thank	#next	#entropy	#hata	#sorun	#anlamıyorum	...	Q_0	Q_1	Q_2	Q_3	Q_4	Q_5
0	0031c86e-81f4-4eef-9e0e-28037abf9883	14.0	3.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.153582	0.294878	0.522162	0.352245	0.460602	0.553879
1	0225686d-b825-4cac-8691-3a3a5343df2b	18.0	0.0	0.0	0.0	0.0	3.0	0.0	0.0	0.0	...	0.192067	0.795607	0.772174	0.882656	0.607118	0.987511
2	041f950b-c013-409a-a642-cffff60b9d4b	9.0	3.0	0.0	1.0	0.0	3.0	0.0	0.0	0.0	...	0.258378	0.295921	0.624849	0.351870	0.643034	0.454330
3	04f91058-d0f8-4324-83b2-19c671f433dc	20.0	1.0	1.0	0.0	0.0	3.0	0.0	0.0	0.0	...	0.146047	0.117834	0.267381	0.316809	0.333889	0.314856
4	089eb66d-4c3a-4f58-b98f-a3774a2efb34	86.0	1.0	5.0	0.0	0.0	26.0	0.0	0.0	0.0	...	0.344280	0.575528	0.782873	0.624833	0.724877	0.872171

Cell 30: Merges two DataFrames, 'df' and 'scores,' based on a common "code" column, cleans the data by removing missing values and duplicates, and produces a sorted DataFrame ('temp_df_sorted_regular') based on grades in descending order.

Cell 31: Shapes of X and y variables are initialized by extracting the features from the 'temp_df' DataFrame.

Cell 32: The ‘train_test_split function’ is applied from scikit-learn. The split is performed with a test size of 20%, and the random state is set to 42 for reproducibility.

Cell 33: Initializes a Decision Tree Regressor with a random state of 0, uses mean squared error as the criterion, and sets the maximum depth of the tree to 10.

Cell 34: Uses the trained Decision Tree Regressor to predict the target variable for the test set (X_test). The R-squared score is then calculated by comparing the predicted values (y_pred) with the actual values (y_test).

- R-squared score: -0.26

Cell 35: Calculates the Mean Squared Error (MSE) and Mean Absolute Error (MAE) between the predicted values (y_pred) and the actual values (y_test) for the regression model.

- MSE: 141.17
- MAE: 6.90

Cell 36: Extracts the Mean Squared Error for each node in the trained Decision Tree Regressor.

Cell 38: Uses the export_graphviz function from scikit-learn's tree module to generate a DOT format representation of the trained Decision Tree Regressor.

- graph.pdf (the output file) can be found in the GitHub repository of the project.

Cell 39: Creates a new Decision Tree Regressor (optimal_regressor) with specific hyperparameter values, such as a maximum depth of 50, maximum features set to 'log2', minimum samples per leaf set to 4, and minimum samples per split set to 5. The optimal regressor is then trained on the training data (X_train and y_train).

- DecisionTreeRegressor(max_depth=50, max_features='log2', min_samples_leaf=4, min_samples_split=5)

Cell 41: Utilizes scikit-learn's GridSearchCV to systematically explore hyperparameter combinations for a Decision Tree Regressor, considering parameters such as max_depth, min_samples_split, min_samples_leaf, and max_features, using 5-fold cross-validation with the negative mean squared error as the scoring metric.

```
Fitting 5 folds for each of 162 candidates, totalling 810 fits
En İyi Parametreler: {'max_depth': 40, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2}
En İyi Skor: 161.62842105263155
```

Cell 42: Fits a linear regression model to the training data, predicts the target variable for the test data, and calculates mean squared error and R-squared for model evaluation.

Cell 43: Trains a Random Forest Regressor with 100 decision trees on the training data, predicts the target variable for the test data, and calculates mean squared error and R-squared as evaluation metrics for the Random Forest model.

```
Random Forest MSE: 114.15411600000002
Random Forest R^2: -0.01682185816492332
```