Homework 5

Exercise 1: What happens in Vegas, stays in Venmo

Venmo is a peer-to-peer (P2P) mobile payment app owned by PayPal, enabling users to send and receive money with just a tap. In 2023, Venmo processed approximately $270 billion in total payment volume, reflecting its significant role in the digital payments landscape. The platform's popularity in the U.S. stems largely from its social features: each transaction includes a message describing its purpose, transforming routine financial exchanges into shared social experiences. This unique blend of finance and social interaction has helped Venmo evolve from a simple payment tool into a vibrant social platform.

In this assignment, you will work with a sample of Venmo's dataset and answer a series of questions using Apache Spark. All data processing and analysis should be performed in Spark. For the social network questions, you may use Python libraries such as NetworKit or networkX, or implement your own user-defined functions (UDFs) in PySpark. For visualizations, you are free to use any Python plotting libraries.

Disclaimer: Do not distribute this homework or the accompanying Venmo dataset to anyone outside of this class.

Important Note: As with any empirical analysis, you will need to make certain assumptions when addressing the questions. Clearly state any assumptions you make in your report for each question. Additionally, ensure that your code is well-commented, explaining your logic and choices throughout. This will not only support the grading process but also enhance your own understanding of the material.

Part 1: Text Analytics (25 points)

Venmo data offers a unique window into what people are spending their money on. However, the biggest challenge lies in having the right lexicon—and most comprehensive lexicons are proprietary, for good reason (as you'll soon discover). Venmo messages can consist of: 1) emoji, 2) text and 3) a combination of emoji and text.

To help you with emoji-based messages, I'm providing an emoji classification dictionary that includes all emoji along with their categories (scraped from emojipedia.com). This should make it easy to classify messages containing emoji.

Text messages, however, are trickier to classify. To guide you, I've created a word classification dictionary, available here:: https://docs.google.com/spreadsheets/d/110FGFJpel_5tpa-UUVXviZ7lMv5akPxOiXUSscdccYQ/edit?usp=sharing. This dictionary organizes words into nine topics: 1) People (usually, words that indicate emotions towards others), 2) Food, 3) Event, 4) Activity, 5) Travel (note that Travel is a subset of Activity, but in my opinion required to be a different topic), 6) Transportation, 7) Utility, 8) Cash and 9) Illegal/Sarcasm (You'll notice several "bad" words—some may be sarcastic, others possibly illicit. My German colleague once got banned from Venmo for using one of them, but that's another story...)

Q1 [5 pts]: Using both the text and emoji dictionaries, classify each transaction in your dataset.

1. What percentage of transactions are emoji-only?
2. What are the top 5 most frequently used emoji?
3. What are the top 3 most common emoji categories?

Answering these questions will give you an aggregate view of Venmo users' transaction profiles. Let's now shift from the aggregate view to individual spending behavior.

Q2 [5 pts]: For each user, create a profile based on their transaction categories. For example: If a user made 10 transactions—5 categorized as "Food" and 5 as "Activity"—their profile would be: 50% Food, 50% Activity.

Q3 [5 pts]: In the previous question, you got a static spending profile, but user behavior evolves. In this question, you'll build dynamic profiles that change over time. Therefore, let's explore how a user's spending profile is evolving over her lifetime in Venmo. Here's how:

- For each user, split their transaction history into **monthly intervals**, starting from month 0 (their first transaction), and track behavior through month 12.

- Example: A user's first transaction is a 🍕 emoji. Her profile at month 0 is 100% Food.

- By the end of her first month, she has made 4 transactions: 2 Food, 2 Activity. Her profile for month 1 is 50% Food, 50% Activity.

- Continue this process up to month 12.
  (*Hint: Window functions will help you here.*)

If you do this right, you will create a dynamic spending profile for each user. However, this is meaningless to plot. So, let's look at this behavior across users and plot the spending profile of the average user. To do this:

- For each month (0 to 12), calculate the average and standard deviation of each spending category across all users.
- Plot time (in months) on the x-axis. On the y-axis, plot the average share of each category, surrounded by a confidence band (±2 × standard deviation).
- What do you observe? Does the average user's spending behavior stabilize over time?

Q4 [10 pts]: **Topic Discovery with LLaMA 3-8B Embeddings**

Recent foundation models can capture rich semantic meanings without hand-curated dictionaries. In this task you will use a pre-trained LLaMA 3-8B model (e.g., the Hugging Face checkpoint `meta-llama/Meta-Llama-3-8B-Instruct`) in Google Colab to embed each Venmo message and discover topics automatically.

1. **Generate message embeddings**
   - Remove emoji so you embed only the text content.
   - Use the model's built-in embedding head (or the mean of hidden states if preferred).
   - Work in **batches** to avoid memory issues; sampling 50 k–100 k messages is fine if the full set is too large.
2. **Cluster the embeddings**
   - Apply an unsupervised algorithm (e.g., k-means, HDBSCAN, spectral) and justify your choice.
   - Determine a sensible number of clusters, using an elbow plot or silhouette scores.
3. **Label and interpret clusters**
   - For each cluster, show the top 10 representative messages or keywords.
   - Give each cluster a short name (e.g., "Food/Drink," "Bills & Utilities").

4. **Compare with my nine hand-built categories**
    - Map each clustered message to its hand-assigned category from Q1–Q3.
    - Report a simple overlap metric (e.g., Adjusted Rand Index or purity).
    - Briefly discuss two pros and two cons of using LLM embeddings versus rule-based dictionaries for Venmo text.
5. **Plotting**
    - Visualize the clusters: reduce embeddings to 2-D with UMAP or t-SNE and color by discovered topic.
    - Place the plot in your report with a concise caption.

Practical Note on Using LLaMA 3-8B (or any gated Hugging Face model)

To use an open-license model from Hugging Face, you must do the following steps:

Step 1 — Accept the License (one-time click)

1. Create / log in to your free Hugging Face account.

2. Visit the model page:
   `https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct`

3. Click “Agree and access” to accept Meta’s Community License.

    - Until you click, the repo is *gated* and will return “Permission required” or HTTP 401 errors. (Hugging Face, GitHub)

Step 2 — Authenticate in Colab

```
!pip install -q transformers accelerate
!huggingface-cli login   # paste your HF API token when prompted
```

or set

```
import os
os.environ["HUGGINGFACE_HUB_TOKEN"] = "hf_XXXXXXXXXXXXXXXXXXX"
```

Step 3 — Load the Model and Tokenizer

```
from transformers import AutoTokenizer, AutoModel
model_id = "meta-llama/Meta-Llama-3-8B-Instruct"
```

```
tok     = AutoTokenizer.from_pretrained(model_id, torch_dtype="auto")
model   = AutoModel.from_pretrained(model_id, torch_dtype="auto",
device_map="auto")
```

(A free Colab GPU can handle small batches if you cast to `float16` and use only the embedding output.)

Part 2: Social Network Analytics (30 points)

Let's now explore a user's social network.

Q5 [10 pts]: Write a script to identify each user's friends and friends of friends. (Definition: A friend is anyone who has transacted with the user—either by sending or receiving money.)

- Describe your algorithm clearly.
- Calculate its computational complexity.
- Can you improve its efficiency?

Q6 [20 pts]: Now that you have each user's list of friends and friends of friends, you're ready to compute various social network metrics. Using the same dynamic framework from earlier, **calculate** the following metrics over the user's lifetime on Venmo, from month 0 through month 12.

i) **Number** of friends and number of friends of friends [very easy, 5 pts].

ii) Clustering coefficient of each user's network [easy, 5 pts]. (**Hint**: The simplest way is to program it yourself. Alternatively, you can use Python libraries like `NetworKit` or `networkX`, though the latter may significantly slow down your script.)

iii) **PageRank** of each user (hard, 10 pts). (**Hint**: You'll need to use GraphFrames for this. Note that PageRank is a global social network metric. If you try to compute PageRank at each point in a user's lifetime, you'll quickly hit a dead end. Can you think of a smarter way?)

Part 3: Predictive Analytics with MLlib (20 points)

If you have survived this assignment so far, well done! Now, let's put all this work into a problem that every company that deals with customers wishes to solve. One of the

biggest questions in Customer Relationship Management (CRM) is whether you can predict in advance how many times your customers will transact. In this subsection of the homework, we will investigate how the different set of metrics that you have created above (text and social) can help us predict the total number of transactions a user will have by the end of their first year in Venmo.

Q7 [5 pt]: Create the **dependent variable Y**, defined as the total number of transactions a user makes during their first 12 months on Venmo.

Q8 [5 pts]: Create the recency and frequency variables. In CRM, this predictive framework is known as **RFM**. Here, you don't have monetary amounts, so we will focus on just **RF**. Recency refers to the last time a user was active, and frequency is how often a user uses Venmo in a month. For each user and each lifetime month (0 to 12), compute:

- **Recency**: Days since the user's last transaction in that month
- **Frequency**: Average number of days between transactions during that month

Hint: For example, if a user had two transactions in their first month, and the last one occurred on day x, then:

- Recency = `30 - x`
- Frequency = `30 / 2 = 15`

Q9 [10 pts]: Using data available at each lifetime point (0 to 12), train three different predictive models to estimate `Y`:

- **Model A:** Recency and Frequency (RF)
- **Model B:** RF + Spending Behavior Profile
- **Model C:** RF + Social Network Metrics

1. For each model, compute and plot Mean Squared Error (MSE) across time. Hint: On the x-axis, plot Lifetime month (0 to 12). On the y-axis, plot MSE for each model.

2. Which model performs best at which stages?
3. Are social or behavioral variables more helpful earlier or later in a user's lifetime?
4. What does this imply about data value and timing for customer prediction?

Exercise 2: Graph Neural Networks (GNNs) in Action (25 points)

The objective of this exercise is to learn how a GNN builds useful representations on a small real-world graph, compare it to a non-graph baseline, and reflect on what you gain relative to manual feature engineering.

We will use the standard Cora citation network (2,708 papers, 5,429 edges, 1,433-dim bag-of-words features, 7 research fields). Load it in Colab with

```
!pip install --quiet torch_geometric torch_scatter torch_sparse
pyg_lib torch_cluster
from torch_geometric.datasets import Planetoid
data = Planetoid(root='.', name='Cora')[0]
```

Q1 Warm-up: Inspecting the Graph [5 pts]

1. Print the number of nodes, edges, feature dimension, and class distribution.
2. Plot the degree histogram on a log-log scale (use matplotlib).
3. Question: What type of degree distribution do you observe and why is this typical for citation networks?

Q2 Baseline Without Graph [5 pts]

Train a logistic-regression classifier on the raw bag-of-words features (no edges). Report test accuracy (use the built-in train/val/test masks). Hints: Use torch.nn.Linear, cross-entropy loss, torch.no_grad() for evaluation.

Q3 Two-Layer GCN [10 pts]

1. Implement a 2-layer Graph Convolutional Network (GCN) with hidden size 16, ReLU, 0.5 dropout.
2. Train for 200 epochs (use Adam, lr = 0.01, weight decay = 5×10^{-4}).
3. Record train, validation, and test accuracies every 10 epochs; plot them.

4. Discussion (≤200 words):
    a. How much does the GCN outperform the no-graph baseline?
    b. Where do you see over-/under-fitting signals in the curves?

Q4 Interpreting Learned Representations [5 pts]

1. After training, extract the 16-dim hidden embeddings for all nodes. Use t-SNE to project them to 2-D and scatter-plot, coloring by true class.
2. Question: Do nodes of the same research field cluster together? Explain what this says about automatic feature learning vs. the manual network metrics you computed earlier in the homework.
3. (**Reflection**) What did we gain? In ≤150 words contrast the **node embeddings** learned by your two-layer GCN with the bag-of-words feature vectors used in your logistic-regression baseline.
    a. Identify at least two concrete advantages the GCN representation offers over the bag-of-words baseline.
    b. Identify the trade-off of the GCN approach