Introduction to Data Science — Assignment 5&6

Instructors: **Dr. Bahrak, Dr. Yaghoobzadeh**

TAs: Hesam Ramezanian, Ali Hamzepour, Mina Shirazi, Reihane Yordkhani, Farbod Azim-mohseni

Deadline: 28th of Khordad

Introduction

This assignment explores advanced topics in modern data science through hands-on tasks that reflect real-world challenges. Spanning multiple domains—including natural language processing, computer vision, and semi-supervised learning—the goal is to build a deeper understanding of both traditional machine learning techniques and the capabilities of foundation models such as large language models (LLMs).

The first part focuses on predicting video game review scores using limited labeled data and large amounts of unlabeled text, introducing students to semi-supervised and active learning strategies. The second part transitions to semantic search in Persian-language Q&A data, guiding students through the development of a search engine powered by embeddings, vector databases, and reranking methods. The third task highlights the limitations of classical modeling techniques by working with the SWAG dataset, where students will use LLMs for complex multiple-choice reasoning—applying in-context learning, fine-tuning, and prompt engineering. Finally, the fourth task dives into unsupervised image segmentation, challenging students to apply clustering methods and evaluate them using metrics like IoU and the Dice coefficient.

Throughout the assignment, students will gain experience in preprocessing, model evaluation, and working with high-dimensional data. More importantly, they will develop the judgment needed to recognize when traditional models fall short—and when to turn to more powerful, context-aware approaches.

Task 1 – Video Game Reviews

The rise of online platforms has caused a huge increase in user-created content, like video game reviews. These reviews usually include short written summaries and a score from 1 to 10. While it's easy to collect the written reviews, getting accurate human-assigned scores for each one takes a lot of time and effort. This project focuses on predicting review scores when there aren't many labeled examples available. It looks at different machine learning methods, including semi-supervised learning (SSL). The goal is to build accurate prediction models that use a small amount of labeled data along with a large amount of unlabeled data, reducing the need for lots of manual work.

Dataset Description

Given a collection of video game review summaries, some with associated numerical scores (1-10) and a larger set without, the task is to accurately predict the numerical score for unseen review summaries. The primary challenge lies in the scarcity of labeled data. The project utilizes two distinct datasets, simulating a realistic scenario with limited annotation budget:

- Labeled Dataset (labeled_reviews.csv): Contains two columns: review_text (textual review content) and review_score (integer,1-10).This dataset represents the scarce, high-quality human-annotated data available for initial model training and validation. Its size is deliberately constrained to simulate real-world limitations in data annotation.
- Unlabeled Dataset (unlabeled_reviews.csv): Contains a single column: review_text (textual review content).This larger pool of data is available without associated scores and will be leveraged by semi-supervised and active learning techniques to enhance model generalization without incurring additional manual labeling costs.

The deliberate imbalance in dataset sizes (small labeled, large unlabeled) is central to evaluating the efficacy of techniques designed for low-resource environments.

1. Text Vectorization (21 Points)

Raw textual data must be transformed into numerical representations suitable for machine learning algorithms. These transformations aim to capture the semantic and syntactic properties of the review summaries.

- **SentenceTransformer (Semantic Embeddings):**
    - A neural network model designed to produce dense vector embeddings for entire sentences or paragraphs. These embeddings are optimized such that semantically similar texts are mapped to proximate points in the vector space.

    This makes them highly effective for tasks requiring a deep understanding of sentence meaning.
  - Implementation (7 Points):
    - Install the sentence-transformers library.
    - Load a pre-trained model (all-MiniLM-L6-v2) suitable for general-purpose sentence embeddings.
    - Compute embeddings for all summaries in both the labeled and unlabeled datasets.
- **Word2Vec (Distributed Word Representations):**
  - A predictive model that learns continuous vector representations for words. It operates by predicting surrounding words given a target word or predicting a target word given its context . Word vectors capture semantic relationships (e.g., "king" - "man" + "woman" = "queen"). Sentence-level representations can be derived by aggregating the vectors of all words within a summary.
  - Implementation(7 Points):
    - Utilize the Gensim library to train a Word2Vec model on the combined corpus of all review summaries (both labeled and unlabeled).
    - After training, compute sentence embeddings for each summary by averaging the vectors of its constituent words.
- **Dimensionality Reduction and Visualization:**
  - High-dimensional embeddings can be difficult to visualize. Techniques like Principal Component Analysis (PCA) help reduce the number of dimensions while preserving as much of the original variance as possible. This makes it easier to visually inspect clusters and patterns.
  - Implementation (7 Points):
    - Perform PCA on the generated embeddings to reduce their dimensionality, and create a scatter plot of the resulting data points. Use color to represent the actual scores (for labeled data) to visualize potential clustering and patterns in the reduced space.
    - If you don't observe any clear pattern or clustering in the scatter plot, explain why that might be the case.

2. Supervised Learning Baselines (17 Points)

Before employing advanced techniques, establish baseline performance using only the labeled data. The nature of the score (discrete integers 1-10) allows for two primary modeling paradigms:

- **Classification Paradigm:**
  - Treats each score (1 through 10) as a distinct categorical class. The model learns to assign a specific class label to each review summary.

    - Considerations: Directly handles discrete outputs but might not explicitly leverage the ordinal relationship between scores (e.g., a score of 8 is "closer" to 7 than to 1).
- **Regression Paradigm:**
    - Treats the score as a continuous numerical value. The model aims to predict a real-valued score, which can then be rounded to the nearest integer for final prediction.
    - Considerations: Captures the ordinal nature of scores and allows for predictions between integer values. However, it might produce predictions outside the valid range (1-10) if not constrained.
- **Implementation (17 Point):**
    - Data Split: Partition the labeled dataset into training (80%), validation (10%), and testing (10%) sets.
    - Model Training:
        - Train a suitable classifier (e.g., Logistic Regression, Random Forest Classifier, Support Vector Classifier).
        - Train a suitable regressor (e.g., Linear Regression, Support Vector Regressor, Random Forest Regressor).
    - Evaluation:
        - Classification Metrics: Accuracy, Precision, Recall, F1-score (macro-averaged for multi-class), and Confusion Matrix.
        - Regression Metrics: Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Coefficient of Determination (R2).
    - Comparative Analysis: Evaluate and compare the performance of classification and regression models to identify which approach is more effective for this task. Select the model that demonstrates superior performance for further use.

3. Semi-Supervised Learning (SSL) Strategies (50 Points)

SSL techniques aim to improve model performance by leveraging the large pool of unlabeled data in conjunction with the small labeled set.

3.1. Pseudo-Labeling

- An iterative SSL approach where a model, initially trained on the small labeled dataset, is used to generate "pseudo-labels" for a subset of the unlabeled data. Only predictions made with high confidence are selected. These pseudo-labeled samples are then added to the training set, and the model is retrained on the expanded dataset. This process can be repeated.
- **Key Concepts:**

    1. Confidence Threshold: A critical hyperparameter defining the minimum prediction probability (for classification) or maximum prediction uncertainty (for regression) required for a pseudo-label to be accepted.
    2. Iterative Refinement: The process can be repeated over multiple rounds, potentially adding more pseudo-labeled samples and refining the model's performance.
    3. Risk of Confirmation Bias: If the initial model makes systematic errors, these errors can be reinforced by adding incorrect pseudo-labels, potentially leading to performance degradation. Careful threshold selection is crucial.
- **Implementation (25 Point):**
    1. Train the best-performing baseline model (classifier or regressor) on the initial labeled training set.
    2. Predict scores for all samples in the unlabeled dataset.
    3. Select unlabeled samples for which the model's prediction confidence exceeds a predefined threshold.
    4. Assign these high-confidence predictions as pseudo-labels to the selected unlabeled samples.
    5. Combine the original labeled training data with the newly pseudo-labeled data.
    6. Retrain the model on this expanded dataset.
    7. Evaluate the retrained model on the held-out test set.
    8. If applying this iteratively improves model performance, repeat steps 2–7 for multiple rounds, observing performance changes.

3.2. Active Learning

Active learning is a machine learning paradigm where the learning algorithm interactively queries a human oracle (annotator) to label new data points. The goal is to strategically select the most "informative" unlabeled samples for labeling, thereby maximizing model improvement with minimal annotation effort.

- **Common Query Strategies:**
    1. Least Confidence Sampling: Selects the unlabeled sample for which the model has the lowest prediction probability for its most likely class (for classification) or highest prediction uncertainty (for regression).
    2. Margin Sampling: Selects samples where the difference between the probabilities of the top two predicted classes is smallest, indicating high uncertainty between competing classes.
    3. Entropy-Based Sampling: Selects samples with the highest predictive entropy, which quantifies the overall uncertainty across all possible class predictions.
- **Implementation (25 Point):**
    1. Start with the initial small labeled training set and train the chosen baseline model.

2. For each active learning round:
    - Compute uncertainty scores (based on chosen strategy) for all samples in the unlabeled pool.
    - Select the top-k most uncertain unlabeled samples.
    - Simulate human annotation: For this project, you will manually assign a score to these selected k samples.
    - Add these newly labeled samples to the training set.
    - Retrain the model on the expanded labeled dataset.
    - Record the model's performance on the test set.
3. Repeat for a predetermined number of rounds (e.g., 5-10 rounds), plotting model performance against the cumulative number of labeled examples.

4. Comparative Performance Analysis (22 Points)

- **Summary of Metrics (5 Point):**
  Tabulate the key evaluation metrics (accuracy, F1-score, MAE, RMSE, etc.) for:
    - The initial baseline model (trained only on the small labeled set).
    - The model after each round of Pseudo-Labeling.
    - The model after each round of Active Learning.
- **ROC and AUC Curves (7 Point):**
  Plot ROC (Receiver Operating Characteristic) curves and compute AUC (Area Under the Curve) scores for each of the above models to assess their ability to distinguish between classes.
- **Learning Curves (5 Point):**
  Plot learning curves showing model performance (e.g., F1-score or MAE) on the test set against the total number of labeled samples (original + pseudo-labeled + actively queried).
- **Discussion (5 Point):**
  Analyze and discuss the effectiveness of Pseudo-Labeling and Active Learning. Which method yielded the most significant performance improvement? Under what circumstances (e.g., specific dataset characteristics, model type) might one method be preferred over the other? Discuss the trade-offs and potential pitfalls of each SSL approach, such as the risk of confirmation bias in Pseudo-Labeling or high labeling costs in Active Learning.

Task 2 - Semantic Search on NiniSite

This project involves working with the PerCQA dataset, which contains around 1,000 Persian-language questions and over 21,000 answers from the NiniSite Q&A forum. The goal is to first perform exploratory data analysis and apply basic NLP techniques such as text cleaning, normalization, and tokenization on the informal Persian text. Then, a semantic retrieval system will be developed to retrieve and rank relevant answers based on a user's query using advanced semantic similarity methods.

1. Preprocessing (20 pts + 5 pts bonus)

Normalize Persian and Arabic characters (4 pts)

- Identify and fix Arabic characters (e.g., "ك" ,"ي") used in place of Persian ones (","ی ک""). Use tools like hazm or parsivar after exploring the data for such cases.
- Look for inconsistent punctuation, extra spaces, or unusual symbols. Clean and standardize them using available text preprocessing tools.

Remove diacritics and unwanted characters (3 pts)

- Persian and Arabic texts may contain diacritics (e.g., "ًِ ", " ّ ", "َ َ ", " ُ ") that aren't useful for most NLP tasks. Remove these symbols to simplify the text. Start by cleaning common ones like the examples above, then explore your dataset to find and handle others as needed.

Tokenization (3 pts)

- Tokenization means breaking text into smaller units like words or sentences, which is a key step in most NLP tasks. For example, turning a sentence into a list of words helps with tasks like text classification or sentiment analysis.You can perform tokenization using libraries such as hazm or parsivar, or any other method you prefer.

Remove stopwords (3 pts)

- Stopwords are very common words (like "برای" ,"که" ,"به" ,"از") that usually don't add meaningful information to text analysis. Removing them helps models focus on more important words.You can use the built-in stopword list in the hazm library or define your own list based on your task.

Stemming and Lemmatization (4 pts)

- Words in Persian often appear in different forms (e.g., "گفتند" ,"می‌گوید" ,"گفت"). Reducing them to a base form helps group similar words and improves model performance. Use hazm.Stemmer() for root extraction or hazm.Lemmatizer() for dictionary forms.

Stemming cuts off word endings to get the root (e.g., "رفت" → "رفتیم").
Lemmatization finds the base dictionary form (e.g., "رفتن" → "رفته‌ام").
This helps reduce vocabulary size and improve model performance.

Normalize informal stretching and repetition (3 pts)

- In casual writing, users often repeat letters for emphasis—like "عاااالیییی" instead of "عالی". These exaggerated forms can confuse NLP models and increase vocabulary size unnecessarily.

  Ask yourself: *Should "عاااالی" and "عالییییی" really be treated as different words?*

- To handle this, use text normalization tools such as hazm or apply regular expressions to reduce repeated characters.

Replace informal or slang expressions(optional - 2 pts bonus)

- Informal words like "خخخ" or "عههههه" are common in social media texts but can reduce model accuracy if not handled properly.

  Replace slang terms with their standard equivalents using a custom dictionary or tools like hazm or manual mapping.

[Optional – 3 pts bonus] Displaying Persian Text Correctly
Persian text may appear with broken or disconnected letters in Jupyter Notebook due to Unicode limitations and lack of right-to-left rendering. To fix this, use arabic_reshaper for shaping and python-bidi for proper RTL display. This ensures that Persian words appear correctly in visualizations and outputs.

2. EDA (20 pts)

Understand the structure of questions and answers: (4 pts)

- First display 2 sample questions along with their corresponding answers.
- Then, compute the average and median lengths across the entire dataset—both by word count and by character count.
- Visualize the distribution of lengths using:
  - Histograms to observe shape.
  - Boxplots to detect outliers and spread.

Identify the most engaging questions: (3 pts)

- Find which questions (`QID`) have the highest number of answers. Analyze overall response rates to understand which topics get more attention.

Analyze user activity patterns: (4 pts)

- Use the `CDate` field to:
  - Determine peak hours and days when users are most active.
  - Identify when the platform is most responsive.Visualize this using Bar plots or line charts for hourly/daily activity.Heatmaps to show intensity of activity across time.

Detect top answer contributors: (3 pts)

- Count the number of answers posted by each user (`CUsername`).
- Create a bar chart of the top contributors (e.g., top 10 users) to show who is most active in the community.

Linguistic and word-level analysis: (6 pts)

- Extract and visualize most frequently used words in both questions and answers.
- Generate word clouds to capture common language patterns and user concerns.
- Go deeper with n-gram analysis:
  - Plot unigram, bigram, and trigram frequencies. Perform this both before and after stopword removal to observe the effect on meaningful patterns.

3. Analyse the dataset (60 pts)

You've recently been hired as a data scientist at Nini Site, a Q&A-based platform. Your first assignment is to improve the site's internal search functionality.

Currently, the site uses traditional keyword-based search algorithms, which are insufficient for retrieving results that are semantically similar to a user's query. To solve this problem, you propose using embedding-based search that captures the meaning of queries rather than just matching keywords.

After researching available options, you choose bge-m3, a multilingual embedding model capable of capturing semantic similarity across various languages. To store and search embeddings efficiently, you decide to use LanceDB, a vector database that is easy to set up and integrates well with modern embedding workflows.

Your job is divided into the following steps:

1. **Load the `bge-m3` embedding model and test it.**
    - Use the model to encode the `QBody` (question body) of a random row from the dataset.
    - Analyze the output of the model: What does it return? What do the components of the output represent? Explain their meanings and potential uses.
2. **Install LanceDB and set up an embedding function.**
    - Install the LanceDB Python library.
    - Use the `TextEmbeddingFunction` class to define a custom embedding function that uses the `bge-m3` model to encode question texts. Only use the dense embeddings.
3. **Define a schema for your vector database.**
    - Your schema must include at least:
        - `qid`: a unique identifier for each entry
        - `qbody`: the raw question text
        - `embedding`: the vector representation of the `qbody` using your custom embedding function
4. **Create and populate a LanceDB table.**
    - Use your defined schema to create a new table.
    - Load the Nini Site dataset (questions only, excluding comments) into this table.
    - Note that the embeddings must be generated automatically by LanceDB using the custom embedding function. You should not compute the embeddings manually before inserting the data into the database.
5. **Perform semantic search with LanceDB.**
    - Use LanceDB's `search` function to retrieve the top 5 semantically similar results for at least 5 different user queries.
    - Evaluate whether the results are semantically relevant to the query Manually.
6. **Implement classical full-text search using LanceDB.**
    - Read the documentation on how to index fields for full-text search.
    - Create a full-text index on `qbody` and perform the same 5 queries from Task 5.
    - Compare the results with those from semantic search. Which approach gives more relevant results?
7. **Research hybrid search techniques.**
    - Investigate and explain what hybrid search is and explain why hybrid search might be more effective than using one method alone.
8. **Evaluation methods.**
    - In this task you have manually checked whether the results were good or not. Research common evaluation metrics for search systems and only explain them briefly in your report (e.g., precision@k, recall, NDCG).

4. Answer Ranking Enhancement with a Reranker (10 pts Bonus)

After performing semantic search and retrieving relevant answers, apply a reranker model to improve the ranking of the results. A reranker evaluates each (question, answer) pair more precisely to determine better relevance ordering.

Example tools:
Use a ready-made model such as bge-reranker or a cross-encoder from the sentence-transformers library.

Steps:

- Perform semantic search for 5 user queries using the embedding-based method.
- Use the reranker to re-rank the top retrieved answers.
- Compare and report the difference in results before and after reranking.

Notes

- The search engine focuses only on the questions/topics in the dataset, not on the comments or answers.
- LanceDB provides an AskAI feature to help you understand its documentation and codebase—feel free to use it if you get stuck.
- You are required to answer questions related to the concepts used in the project, which are part of the 60-point third section of this task.

Task 3 – LLM for MCQs

In data science, not all problems fit traditional models like classification or regression. Tasks involving nuanced language or semantic understanding require more than pattern recognition. In such cases, large language models (LLMs), with their ability to grasp context and intent, are essential. Data scientists must know when to move beyond conventional models and apply LLMs for deeper comprehension.

To understand the limits of traditional methods and the importance of semantic reasoning, you'll work with the SWAG dataset—over 113,000 multiple-choice questions based on real-world scenarios. This task will help you explore how LLMs excel at complex inference beyond simple patterns, reinforcing the value of foundation models in practical data science.

1. Hugging face access token

A Hugging Face access token is like a password that grants you permission to use Hugging Face services programmatically. It's a secure way to authenticate yourself when interacting with the Hugging Face API, model hub, or datasets.

You will need this for this project, so let's go through the process of creating one:

1. Go to https://huggingface.co and sign in or create a free account.
2. Create an account if you don't already have one. If you do, just login.
3. Click on your profile picture in the top-right corner.
4. Navigate to Settings → Access Tokens.
5. Click New Token.
6. Give it a name like SWAG_Project_Token.
7. Set the scope to write (since we are going to fine-tune a model later).
8. Click Generate Token and copy the token to your clipboard.
9. Your token should look like something like this: "hf_XXXXXXXXXXXXX"
10. You can use this token in your notebook one of the following ways:

```
from huggingface_hub import notebook_login, login

notebook_login()
login(token="hf_XXXXXXXXXXXXXXX")
```

We highly recommend using Kaggle Notebooks instead of Google Colab. Kaggle provides more reliable GPU access, fewer interruptions, and a smoother experience when working with large models.

2. Loading the dataset (2 points)

Begin by familiarizing yourself with the SWAG dataset through its official documentation on the Hugging Face Datasets Hub. For this task, you are required to load the standard version of the SWAG dataset into your notebook using the Hugging Face datasets library. This version includes a rich set of multiple-choice questions designed to evaluate contextual and commonsense reasoning.

3. Analyse the dataset (7 points)

Carefully review the official documentation of the SWAG dataset to gain a clear understanding of the meaning and purpose of each column. This foundational knowledge will enable you to more accurately interpret the behavior and results of language models applied to this dataset. (Do some minor EDAs)

4. Preprocess the dataset (16 points)

Write a preprocessing function that prepares the SWAG dataset for input into a language model. Your function should:

1. Replicate the sent1 (context) field four times—once for each of the four endings.
2. Combine the replicated context with sent2 to complete the sentence setup.
3. Concatenate the result with each of the four candidate endings (ending0 to ending3) to create four full input sequences per example.
4. Flatten the sequences and pass them through a Hugging Face tokenizer.
5. Unflatten the tokenized output so that each example retains a list of four candidate encodings.

This structure ensures that each data point is aligned with the four-way multiple-choice format expected by models like BERT or RoBERTa in multiple-choice classification tasks.

5. Load a tokenizer

To prepare the text inputs for your model, you now need to load a BERT tokenizer from the Hugging Face Transformers library. This tokenizer will be responsible for converting your sentence pairs (the contextual sentence starts and each of the four candidate endings) into token IDs that the model can understand. Use this version of the BERT tokenizer:

```
from transformers import AutoTokenizer
tokenizer =
AutoTokenizer.from_pretrained("google-bert/bert-base-uncased")
```

6. Apply the preprocessing function (5 points)

Apply your preprocessing function to the entire SWAG dataset using the dataset map method.

💡 **Hint: Set batched=True to enable batch processing and improve performance.**

7. Padding (5 points)

In natural language processing tasks, especially those involving multiple-choice questions like SWAG, different examples often have varying lengths. Padding all sequences in the dataset to the maximum possible length would be wasteful, leading to unnecessary computation and memory usage—especially when most sequences are much shorter.

By dynamically padding each batch to the length of the longest sequence within that batch, we ensure that each batch is as efficient as possible, without excessive padding. Hugging Face's DataCollatorForMultipleChoice automates this process for multiple-choice tasks. It:

- Flattens the input data,
- Pads the sequences to the appropriate length within each batch,
- And unflattens them so they can be fed directly into a model like BERT for multiple-choice classification.

8. Load the model

Load the model using the following code:

```
from transformers import AutoModelForMultipleChoice, TrainingArguments, Trainer

model = 
AutoModelForMultipleChoice.from_pretrained("google-bert/bert-base-uncased")
```

9. Test the model on the dataset (10 points)

Using the validation split of the SWAG dataset, evaluate the model's ability to answer multiple-choice questions. Your task involves two parts:

1. Test Case Analysis:
   Select a single example from the validation set and pass it through the model. Carefully examine the model's predicted answer against the ground truth label. Discuss the correctness of the prediction, and reflect on any reasoning patterns (if discernible) from the model's output.

2. Comprehensive Validation Set Evaluation:
   Systematically process the entire validation set through the model. Define and compute at least one metric to quantify the model's performance across all examples.

10. Use in-context learning (ICL) (20 points)

In-Context Learning (ICL) is a method where a model is given examples of tasks directly within its input prompt, enabling it to learn from these examples without updating its weights. Instead of fine-tuning the model, we provide a few example question-answer pairs, and the model uses them as guidance to make predictions on new questions.

Your goal is to modify the model's input format to include a few-shot setup using ICL. In-Context Learning (ICL) techniques have diversified over time, enabling models to generalize across tasks by conditioning on examples provided in the prompt. One of the foundational works, *Language Models are Few-Shot Learners*, introduced the idea of few-shot ICL, where models adapt to new tasks by observing a few examples at inference time without fine-tuning. Building on this, *Large Language Models are Zero-Shot Reasoners* demonstrating that prompting models with "Let's think step by step" can unlock reasoning abilities in a zero-shot setting. *Rethinking Few-Shot Learning* analyzed the underlying mechanisms of ICL, revealing that models often rely on pattern recognition rather than true understanding. InstructGPT (*Training Language Models to Follow Instructions*) introduced instruction tuning, showing that aligning models with human intent improves ICL performance. Finally, *Transformers Learn In-Context by Gradient Descent* provided a theoretical perspective, suggesting that models simulate a form of gradient descent internally during ICL, framing it as a process of implicit learning.

You should choose 2 of the techniques mentioned above and process the validation set using your chosen ICL format and evaluate the model's performance again. Compare the two methods with each other and with the baseline model (without ICL).

Evaluate the model's prediction accuracy on the validation set under each scenario and report your findings clearly in a table.

Keep your implementation simple—focus on adapting the prompt structure and observing how the model's predictions change.

11. Fine-tune bert (30 points)

Now that you have your model set up, your next task is to fine-tune it using the SWAG training set.

What is Fine-Tuning?

Fine-tuning is the process of taking a pretrained model (like BERT) and continuing to train it on a specific task or dataset. While BERT is trained on large amounts of general text data, it has not seen the specific format or reasoning patterns required by the SWAG dataset. By fine-tuning, we adjust the model's internal weights to better align with the structure and distribution of the SWAG data, helping it learn to predict more accurate answers for this particular task.

Why is Fine-Tuning Useful?

Without fine-tuning, the model relies purely on its general knowledge, which might not perfectly match the SWAG task. Fine-tuning allows the model to:

- Adapt to the specific question/answer format in SWAG.
- Learn common patterns in the contexts and endings of the dataset.
- Improve performance beyond what is achievable with zero-shot or in-context learning alone.

For this project, you will fine-tune your BERT model using **LoRA (Low-Rank Adaptation)**, a powerful, parameter-efficient fine-tuning method.

What is LoRA?

LoRA (Hu et al., 2021) introduces small, trainable low-rank matrices into the model's architecture, allowing you to adapt the model to a new task without updating all of its parameters. This dramatically reduces the memory and computational costs of fine-tuning large models like BERT.

Using LoRA enables you to:

- Fine-tune large models efficiently, even on limited hardware (like Kaggle GPUs).
- Save time and memory compared to full model fine-tuning.

Gradient Accumulation

When working with large models and limited GPU memory, it's common to use gradient accumulation. Instead of updating the model after every mini-batch, you accumulate gradients over multiple smaller batches and update the weights after a larger "virtual" batch.

For example, if you want an effective batch size of 64 but can only fit 16 samples in memory, you can accumulate gradients over 4 steps before performing an update.

Here’s how to proceed:

1. **Prepare the Data**
    - Use the SWAG training set (split="train") for training and the SWAG validation set (split="validation") for evaluation.
    - Apply your existing preprocessing pipeline to the data. Ensure that the model inputs—context, question, and multiple endings—are properly formatted.
2. **Fine-Tuning the Model**
    - Fine-tune the model using LoRA on the SWAG training set (split="train") for at least two epochs (feel free to train longer).
    - Use gradient accumulation to simulate a larger batch size.
    - Set a reasonable batch size per step (e.g., 8 or 16).
    - Set the learning rate (e.g., 2e-5, 1e-5, 2e-3).
    - Optionally, experiment with learning rate scheduling or early stopping for improved results.
3. **Evaluate the Model**

    After training, evaluate your model on the SWAG validation set using the following three key metrics:

    - Accuracy: The percentage of correctly predicted answers. This is your baseline performance measure for classification tasks.
    - Perplexity: Perplexity measures how well the model predicts the correct output, given the input. Lower perplexity indicates the model is more confident and accurate in its predictions. This metric is especially useful in language modeling tasks and for evaluating LLMs.
    - Confusion Matrix: A matrix showing how often each class is predicted correctly or misclassified. This helps identify specific patterns of errors and which classes the model struggles with.
4. **Report Your Results**
    - Provide your final validation accuracy.
    - Compare the fine-tuned model’s performance to your earlier results from zero-shot and in-context learning setups.

This fine-tuning step allows the model to directly learn from the SWAG dataset, adapting its weights to better predict outcomes based on the data distribution.

12. In-Context Learning with the Fine-Tuned Model (10 points Bonus)

Now that you have fine-tuned your model on the SWAG dataset, let's explore how combining fine-tuning and in-context learning (ICL) affects performance.

Here's what you need to do:

- Apply the same ICL setup you used earlier (with a few-shot prompt of example question/answer pairs) but now use your fine-tuned model instead of the original, unmodified model.
- Evaluate the model's performance on the validation set again using the same metrics (accuracy, per-class accuracy, etc.).
- Compare the results to your earlier evaluations from:
    - Zero-shot (pretrained BERT, no fine-tuning, no ICL)
    - ICL with the pretrained model (before fine-tuning)
    - Fine-tuned model (without ICL)

13. Analyse the results (5 points)

You've now tried three approaches: fine-tuning, in-context learning, and combining both. Which one gave you the best results—and why do you think that happened?

For example, you might consider:

- Does the fine-tuned model still benefit from ICL, or is the effect less significant now?
- Could there be cases where ICL introduces noise or distracts a model that's already specialized on a task?
- In what kinds of real-world scenarios would combining fine-tuning and ICL be especially useful?

Questions

1. Fine-tuning a large model like BERT can be powerful but also resource-intensive. In what scenarios might fine-tuning be the best approach? When might you prefer ICL instead? Are there cases where you might combine both?
2. In your ICL prompts, you used examples from SWAG. If you had to design a custom ICL prompt for a different task (e.g., predicting medical diagnoses or financial trends), what kinds of examples would you choose? What factors would you consider when selecting them?

Task 4 - Image Segmentation Using Clustering Methods

Image segmentation is one of the most classic and extensively studied tasks in computer vision, with a wide range of applications—from segmenting tumors in medical images to identifying and separating different elements of a road scene in autonomous driving, such as lanes, vehicles, pedestrians, and traffic signs. Despite this, image segmentation tasks often face substantial challenges related to data. One of the primary issues is the need for high-quality, pixel-level annotations, which are both time-consuming and expensive to produce—especially in domains like medical imaging, where expert knowledge is required. To address these challenges, unsupervised methods have been deployed to automatically generate segmentation masks required for training these data. One such method is clustering, which groups pixels based on their features—such as color, intensity, or spatial information— into different segments. Although this might be a complex problem in cluttered images it is possible for some image datasets. In this section you are going to create segmentation masks for players from the football player segmentation dataset.

0. Understanding Segmentation

The goal of image segmentation is to divide an image into meaningful parts or regions. This is done by assigning a label to each pixel. Note that this is different from classification where each image is assigned to one or multiple labels. This makes segmentation essential for tasks requiring precise localization, such as identifying individual players in a football game or isolating specific objects in complex scenes. Segmentation tasks can be categorized into multiple categories.

Segmentation tasks can be categorized into multiple categories. The simplest one is **semantic segmentation**, which assigns each pixel a class label based on the object or region it belongs to, without distinguishing between different instances of the same class. For example, in a football game image, all pixels belonging to players would be labeled as "player," regardless of which specific player they represent. **Instance Segmentation** is another type of segmentation whose goal goes beyond semantic segmentation by not only labeling pixels by their class but also differentiating between individual instances of the same class. In the football player dataset, instance segmentation would assign unique labels to each player, enabling separation of player 1 from player 2, even though both belong to the "player" class. Finally **Panoptic Segmentation** combines semantic and instance segmentation, labeling every pixel with both a class and, for countable objects, an instance ID. In the context of the football dataset, panoptic segmentation would assign unique IDs to each player while labeling background elements like the ground or crowd as single, separated regions. Focus of this project is mainly on semantic segmentation. You are going to use simple clustering approaches for this assignment. The reason is that it is possible to cluster image pixels where each cluster becomes one segment of the image. With this knowledge you are ready to start the next parts. Good luck!

1. Dataset Loading

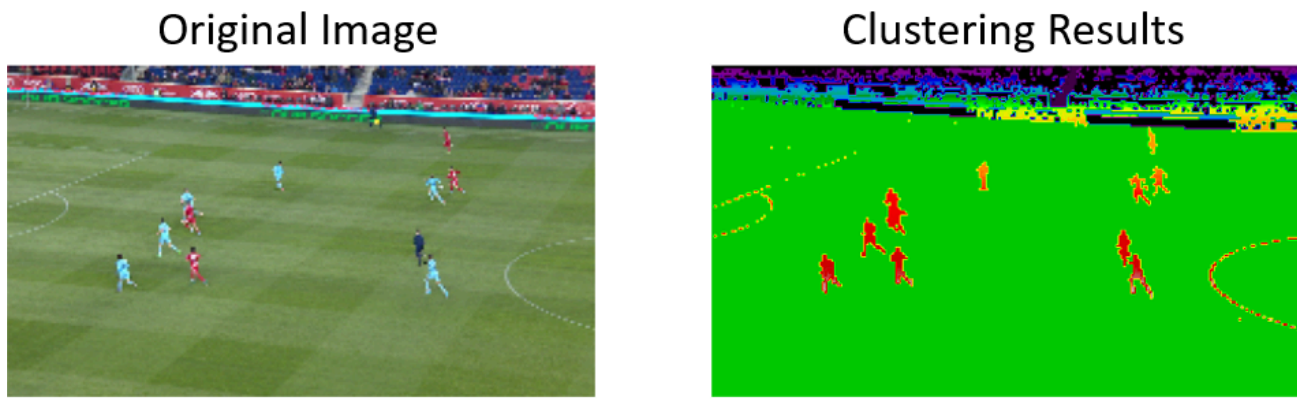
- Load the dataset from the Kaggle link. Because this section contains no training and not the whole dataset is needed. Sample 50 images from the dataset and use them for the rest of this section. This dataset contains two directories. The annotations which contain a json file of all segmentation masks and the images. The images are in size of 1920*1080. Down scale images to 1/8 or 1/16 of the original size or otherwise you will run into performance and memory issues in the upcoming parts.

2. Creating Features (20 points)

Feature creation is the most important part of the clustering. Pixel features should be crafted in such a way that meaningful regions which include players get allocated into one or multiple clusters. Like the example in the course lectures, it is possible to do the clustering only using pixel colors. But note that this simple feature will not likely result in great results. You should try different more complex feature selection methods and report the results for each of them. For example one such example can be concatenating the pixel colors with the pixel positions in the image. Report and evaluate all feature selection methods you explored.

3. Cluster pixels (30 points)

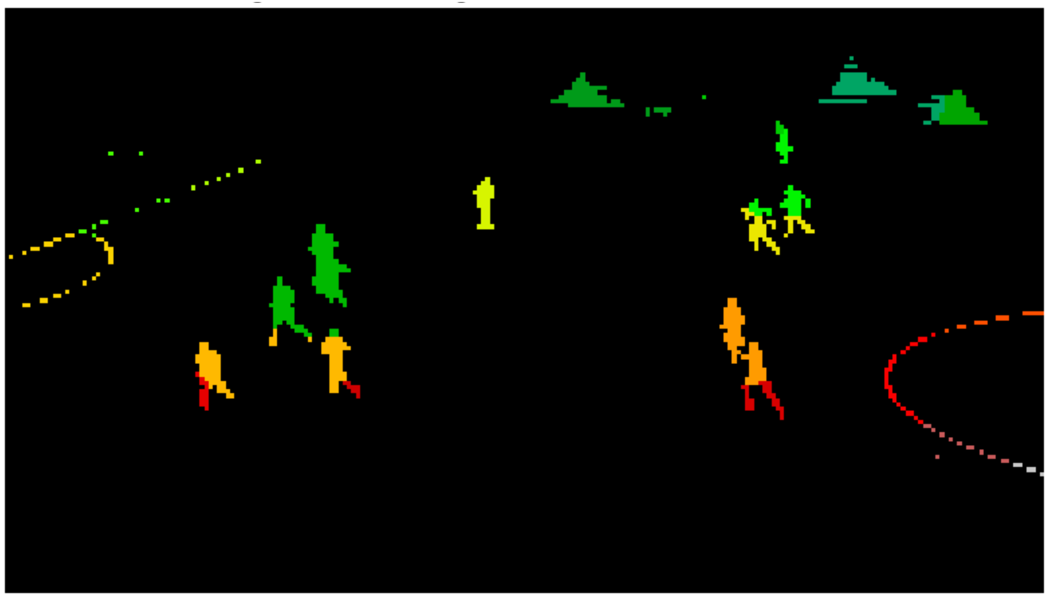
Many different clustering methods from simplest ones like K-Means to more advanced approaches such as density-based clustering (e.g., DBSCAN) and hierarchical methods (e.g., Agglomerative clustering). Use these methods or any similar clustering approach that you find more suitable. The most important task of this part is tuning the clustering parameters. For example if you use K-Means, the selection of K can be challenging especially in this task where it might be difficult to exactly determine the K based on prior knowledge. Use metrics like Silhouette score, inertia, and any suitable metric you find. Output of this part should be multiple clusters. It is not necessary that each player is clustered in only one cluster. You can visualise your clusters to see the results of your segmentation. Note that you might see that each player is segmented into multiple clusters which is fine up to this part. But be careful that there might exist a problem if for example parts of the ground and a player parts are in the same cluster. An example of the clustering can be like this:



4. Filtering and Merging(15 points)

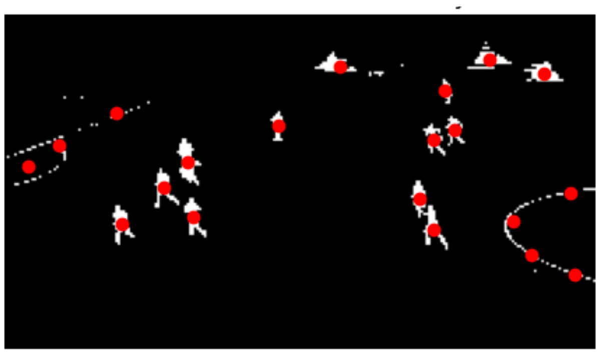
More than the problem mentioned in the last part, there is the problem of segments that do not contain the players, for example the ground. To filter these segments you can set different thresholds on the size of each cluster and apply filters that can drop unrelated clusters. After that you need to merge smaller clusters that are positionally next to each other in the image to create bigger clusters.

After this part you should notice each player is segmented into one or two clusters. An example is shown below.



5. Even More Clustering! (15 points)

In this step, you will generate a binary mask from the clusters obtained in previous parts, where each pixel is assigned a value of 1 (True) for regions corresponding to players and 0 (False) for non-player regions, such as the ground or other background elements. The binary mask should isolate players, ensuring that only their pixels are marked as 1, while all other areas (e.g., the field, crowd, or other objects) are marked as 0. After creating the mask, apply clustering again to identify connected components within the mask and compute their centroids. Finally, visualize the results to display the binary mask and the identified components with their centroids. An example of such a binary mask is shown below. White color corresponds to value 1.



6. More Advanced Features! (10 points bonus)

Use a pretrained convolutional model like ResNet or HRNet to create rich feature representations from the input image. Note that these models decrease the height and width of the model at each step. Down scaling the input image can result in losing details of the image and unacceptable results. I suggest that after generating the features apply DBSCAN and try to tune the parameters so that DBSCAN assigns -1 labels to players pixels.
Explain why DBSCAN shows this behaviour in this part. Finally repeat parts 3 and 4 to get you final player segments.

7. Evaluation (20 points)

To evaluate your segmentations, use the ground-truth segments provided in the JSON file located in the annotation folder of the football player segmentation dataset. For evaluation, create a binary mask from your clustering results (from part 5), where pixels corresponding to players are assigned a value of 1, and all other pixels (e.g., ground, crowd) are assigned 0. Similarly, load the ground-truth annotations from the JSON file and convert them into binary masks in the same format, with player pixels set to 1 and non-player pixels set to 0. The goal is to measure the quality of your segmentation by comparing your predicted binary masks against the ground-truth masks. What we want to measure is the amount of overlap between the output segments and ground truth labels. In this part, we will use the Dice coefficient and Intersection over Union (IoU) as evaluation metrics. Compute and report these metrics on a sample of 50 images from the dataset.

Dice Coefficient:

The Dice coefficient is used to measure the similarity between two sets. In the context of image segmentation, it measures the overlap between the predicted segmentation and the ground truth. It is calculated as twice the area of intersection divided by the sum of the predicted and ground truth areas. A Dice score of 1 indicates perfect agreement, while a score of 0 indicates no overlap.

Intersection over Union (IoU):

Intersection over Union (IoU), is a common evaluation metric for segmentation tasks. It is defined as the area of overlap between the predicted and ground truth masks divided by the area of their union. Unlike the Dice coefficient, IoU gives more penalty to mismatches and is often considered a stricter metric. A higher IoU value shows better segmentation performance, with 1 representing a perfect match and 0 indicating no overlap.

$$IoU = \frac{|A \cap B|}{|A \cup B|}$$

Questions

- Provide an example of a segmentation task where each type of segmentation—semantic, instance, and panoptic—is appropriate and sufficient, and explain why each type suits the specific task.
- Explain the difference between the Dice coefficient and Intersection over Union (IoU) as evaluation metrics in image segmentation. In what scenarios might one be preferred over the other?
- Imagine a scenario where we need to cluster a large set of images into multiple categories. Methods such as K-means can become computationally expensive and inefficient when applied directly to high-dimensional image data. One approach to address this issue is to use an autoencoder architecture to reduce the dimensionality of the images before clustering. Explain how autoencoders can be used for this purpose. How does this method help improve the efficiency and effectiveness of clustering algorithms like K-means on large image datasets?

Notes

- Upload your work as a zip file in this format on the website: DS_CA5_[Std number].zip. If the project is done in a group, include all of the group members' student numbers in the name.
- Only one member must upload the work if the project is done in a group.
- We will run your code during the project delivery, so make sure your results are reproducible.