Project Proposal

Grant Doan, Prayash Joshi, Reagan Orth, Ved Patel

March 3, 2023

1 Technical Description

1.1 Background

Automated recommendation has recently grown in popularity with from multitudes of systems ranging from shopping websites to media distributors and having such a system integrated on modern item cataloging services is an expectation. In the context of movies, these tools help users filter out content that is not relevant to them considering attributes such as the user's watch history, movie ratings, or the content consumed by similar users of the platform. All relevant movie and media content providers deploy these systems to prevent information overload and complex decision-making minimizing the amount of time the user has selecting which item to consume. These systems attempt to predict a rating a user may give an item across a user's dynamic taste in movies, constantly accommodating for changes in the user's preferences. Building a system that accomplishes this effectively and efficiently is the subject of much current research and a variety of approaches in how to select items closest to a user's preferences is still a topic of debate. We discuss one such approach here.

1.2 Problem

Creating accurate systems that can accommodate for the entirety of a user's interest is a feat that can be broken down to fundamental concepts that are applied to broader filtering algorithms which form the baseline to recommendation systems. The way we find an item that falls in line with something a user would consume requires the model we deploy to learn similarities.

One way a machine explores similarity is through item similarity which would look like movies with like attributes (genre, director, actors, theme, plot). It easily perceives quantitative numbers like run time. For qualitative data, the model can create unique labels for attributes that are categories such as genre or director. For more complex categorical data such as plot and script content, natural language processing is deployed to find likeness within documents. These examples of item similarity lends itself naturally to content-based filtering which scores similarities between items by the content-itself. Therefore, if a user were to have a heavy preference watching superhero dystopias, our model would recommend other super hero movies or dystopias. The same could be said for users who particularly enjoy film by a particular director, i.e. a user who likes Tarantino movies is going to be recommended the new Tarantino movie that hit the platform or previous Tarantino movies that the platform has already.

Instead of finding similarities between content that is consumed, collaborative filtering systems finds similarities between users. This is propagated by the idea that a user who enjoys a particular range of films and help similar users expand their horizon of movies if they are

not already watching those types of movies already. A user can be similar to another simply by enjoying the same films creating a relationship between the 2 users. The system can then recommend something that is liked by one user to another user. For example, if user A and user B rate a murder mystery highly, the machine will link the 2 users as similar. When user B then finds a documentary as enjoyable as a murder mystery, the machine will recommend that documentary to user A creating a recommendation to user A from user B. Scaled up to database of user's the size of Netflix, a machine can create networks of similarities that are constantly recommending to a user's dynamic behaviors and changing tastes.

Finally, hybrid filtering combines these two methodologies resulting in a recommender that uses both content and user similarities to match items to users. Therefore, user A, who is similar to user B, is recommended items from user B's taste, similar to the items user A enjoys. Our contribution to this space is an example of collaborative where we find similar items based on the contents of the movie and find similarities between users through their reviews.

Similar to all dimensions of modern computer, data science, and machine learning research, a variety of difficulties present themselves when creating efficient systems that can accurately gauge the taste of a vast number of user's on a media platform.

Cold start comes as a result of the necessity for enough data to get the system started. Having one or two users for examples for the system to learn is simply not good enough. This can come as a result of not having enough users to find similarities between or not having enough movies to find similarities between.

Data sparsity can be applicable to both content and collaborative filtering. This is present when trying to compare 2 users tastes as user A may have ratings for x amount of movies and user B may have ratings for y amount of movies. While they can have ratings for a common movie, there is still a lot of wasted space where the model would store that user A does not have a rating for movie M, but user B does. Scaled up to millions of users and there is a lot of empty space. Similarly, if a model is comparing the scripts of 2 movies with TF-IDF, there is significant empty space between the 2 documents as most the significant words in one script are likely not present in another script. Already being a high dimension vector with the amount of words present in a script, sparsity will only make space issues worse and worse.

Scalability is an issue that occurs due to the need to have more and more data to make our model more accurate. While this is not a unique issue to modern computer science, it is still relevant to the performance of systems.

Security is an ethical concern that exists when a user is suspect of a machine tracking their viewing data and using it without the knowledge of the user. While things like ratings and reviews are made public for other users to explore, users can be more skeptical when a model is tracking their extensive viewing history without them knowing.

While each issue is more or less relevant considering the type of recommendation system a services or user chooses to deploy, they still remain a significant obstacle in progressing highly accurate and efficient performance of modern recommendation.

1.3 Literature Review

A survey of literature reveals efforts to contribute to the space of recommendation systems using the filtering methodologies mentioned above. Although there is rich practice polishing each of these filtering schemes, we turn our attention specifically to papers that focus on hybrid filtering. Furthermore, because our system leverages natural language processing to classify reviews in to contribute to a recommendation system, document similarity and sentiment analysis in recommendation systems were particularly sought after.

In their paper focused on tourism recommendation base on semantic clustering and sentiment analysis, Moud, Nejad, and Sadri explore user preferences in simplifying user decision-making for tourism based on other user reviews from social networks. Each of these reviews runs through a processing pipeline that connects and scores the comments relative to each

other and on whether it is in line with what a user is looking for. Their pre-processing stage leads with tagging scraped reviews into their parts of speech, then taking out words that do not contribute to any meaningful draw of insight. This makes the review more processable for information gain. They then use wordnet in order to map words to their stem or base word. The example the authors use for this is catlike being mapped to cat. After this sentence has been converted, they extract the nouns and vectorize them based on which nouns appear frequently. Upon comparison with other reviews, the authors analyze frequent words that are mentioned as attractions and combines them with other contextual information such as visiting time, weather, and location to create recommendations. Sentiment analysis in this context contributes to the user preference part of the pipeline that adds to contextual information and attempts to bolster the accuracy of attraction recommendation as well, so we have this text processing pipeline contributing to analyzing what the user prefers and the general attractions of the area. While the subject matter of the data is different than what we are looking to analyze for our project, the practice of processing reviews into finding what is generally attractive or not attractive about certain movies is, and discovering how to sift our text data for important keywords is going to improve both the efficiency and effectiveness of our recommendation system.

Yongfeng Zhang also explores sentiment analysis on reviews for recommendation in his paper from Tsinghua University. His exploration of text reviews emphasizes the importance of how they both contain information about the product and about the user and focusing on analyzing phrases in concluding a general opinion on the product for a user. Zhangs processing of textual processing breaks down phrases into nouns and opinions. He also pays special attention to reversal or negating words to catch when a positively connotated word follows a word that negates it. Therefore, an example of a way a review could be broken down follows "Screen is perfect, but the earphones are not that good" being broken down into tuples: (screen, perfect)[normal] and (earphones, good)[reversed]. Zhang uses textual reviews to construct profiles for users, which are formulated tuples across vectors of users for vectors of items. These profile attributes can then be put together to push focus on collaborative filtering, finding users that have similar ratings to be mapped together. To our project, this piece is important to finding ways to map general sentiments on fill attributes to unique users in a way that is does not scale outside our computing power. In creating general sentiments about genres, directors, or other film categories, we drastically cut down the storage and sparsity of matrices that would be present if we try to create vectors for a users general rating of the movies they've seen or rated.

A collaborative filtering approach is suggested Jeong and colleagues that uses different personalization techniques to approach both user based filtering and item based filtering. Their system takes in a user and a user's rating as well as a recommended recipient user. The system they have builds a profile for user N based on their ratings and if their rating, projected and actual, reaches a certain value, they are mapped to a cluster where they are classified into a certain group of users similar to them. Item similarity was also explored in a unique way as they have categories that the movie is mapped into. The categories for each movie were selected based on a preset of what a user selected was important to them. Additionally, combinations of these movies helped create more unique subcategories that could closely identify with a user's preference. For example, a user may prefer age or the age of the intended audience of a movie important. Movies are then chooses arbitrarily as belonging to a category with child main characters or movies created for kids.

1.4 ML Techniques

For this project, there are two current implementations that we are considering. The first is a Long Short Term Memory (LSTM) Recurrent Neural Network. General RNNs are good for many tasks such as natural language processing and speech recognition, but tend to "forget"

old data in favor of the new. For our purposes, there is no reason that new data should take precedence over old, and so we will use an LSTM to attempt to preserve all relevant information and only "forget" information that is not useful to us. LSTMs have been shown to be successful in recommendation problems in the past [4], and therefore can likely perform well in this task.

Our other possible implementation is a Graph Neural Network (GNN). Where LSTMs are good for recommendations in a more conventional machine learning sense, Graph Neural Networks are excellent at modeling relationships [5]. This can essentially be thought of as a large web of nodes that represent the movies, with unweighted edges connecting similar movies. Nodes that are further away in the graph are less similar, nodes that are very close will be very similar. To predict a movie that a user will like, we can check their previous favorite movies in the graph and determine a recommendation based on the results.

1.5 Dataset

The premise of our project is that movie recommendation is best done by using what people have already said about other movies. Therefore, the majority of our data will be web-scraped movie reviews from a large and diverse set of movies written on IMDb. We will include the writer of the review and their rating (out of ten), in order to learn similarity between movies based on reviewer preferences. To supplement this, we will also include other critical movie information, such as the release year, the genre(s), the director(s), some subset of the actors, and possibly the writer(s).

In order to preprocess the review data, most of what we will be doing is typical NLP preprocessing: parsing out the web-scraped HTML, filtering out spaces, punctuation, and capitalization, and using TF-IDF to create a sparse matrix. On top of this we will also increase the importance of actor/director/character names, as these are known to carry strong meaning. Strong semantic words may also receive the same treatment, as any insight into how much the user enjoyed the movie is extremely important for us. Other words will be removed because they are far too common to give us any information. The rest of the words will be included in order to get a sense of what kind of movie it is.

1.6 Evaluation Metric

In order to evaluate our data, we will be using a test set. Unfortunately, movie recommendation is difficult because there are so many movies in existence, and we do not have the ability to ask IMDb users to watch and review movies for us that they haven't seen. Therefore, we are largely limited to what is already in existence online. For many reviewers that we find in our dataset, we will hold back some number of movies they reviewed, and use our model to predict ratings on the held-back movies. If we can accurately predict a user's ratings for arbitrary films, then it is fairly trivial to find a film that the user should rate highly. As additional confirmation, we will also be testing for ourselves and any interested friends: we will input a list of movies and ratings, obtain recommendations from our model, and watch the movies.

2 Project Management

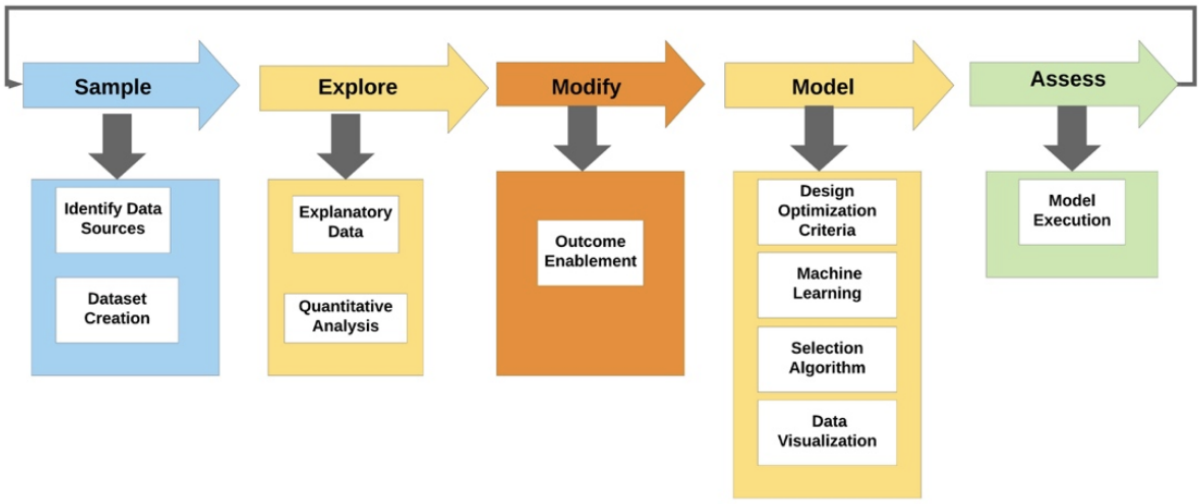
2.1 ML Life-Cycle Methodologies:

There are two main approaches to project management being conisdered for this project: SEMMA methodology and CRISP-DM methodology.

2.1.1 SEMMA Methodology:

The SEMMA methodology is defined by the SAS Institute as a process for discovering previously unknown patterns within large amounts of data that can be used to gain a competitive advantage.SEMMA stands for sample, explore, modify, model, and assess.Unlike other methodologies, SEMMA's steps are not iterative and concentrate solely on procedures rather than outcomes. [3] Each step in the SEMMA methodology is described below:

1. Sample: Collect and Analyze sample data in its original form.
2. Explore: Identify outliers, patterns, and relationships.
3. Modify step: The data is modified by selecting, transforming, and deriving the features required to achieve the desired result.
4. Model: Data analytics algorithms and tools are used to model the data and produce results.
5. Access: The resulting outcome is evaluated in multiple stages by evaluating the usability and reliability of the data mining findings.



2.1.2 CRISP-DM Methodology:

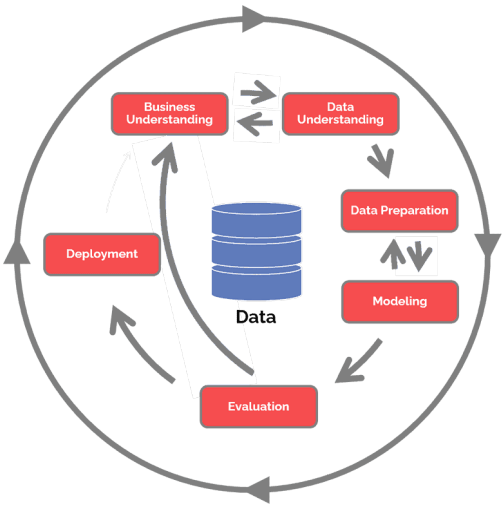
The CRISP-DM approach (CRoss-Industry Standard Process for Data Mining) is widely used by data mining experts. Daimler Chrysler (then Daimler-Benz) first introduced it in 1996. Six iterative phases comprise the CRISP-DM methodology. While the CRISP-DM approach was not designed specifically for deep learning, it can still be a useful framework for guiding the development of deep learning models while keeping business objectives and end users in mind. [3] Each step in the CRISP-DM methodology is described below:

1. Business Understanding: Determine business objectives, assessing the situation, establishing data mining goals, and developing a project plan.
2. Data Understanding: The preliminary data is made accessible for exploratory analysis and data quality assessment.
3. Data Preparation: A comprehensive process where individual steps for feature extraction, data cleaning, data reduction and transformation is executed. This is crucial for the modelling phase which comes next.

4. Modeling Phase: Based on the understanding of all the previous phases, a machine learning model is selected to tackle the specific problem.

5. Evaluation Phase: Results are processed and carefully analyzed. A review is done to make sure it meets the deliverables of the specific problem.

6. Deployment Phase: Plan for deploying, optimizing, scaling, monitoring and maintaining the model.



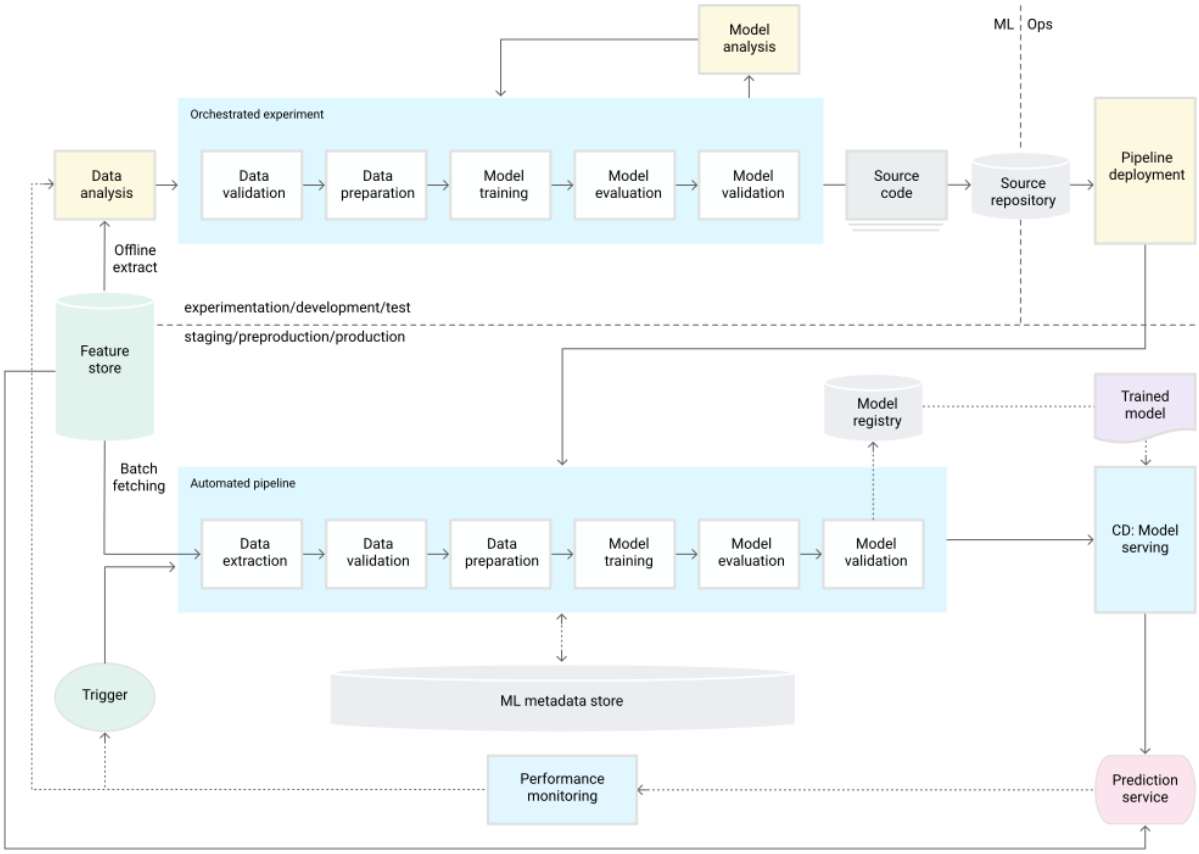
2.2 Continuous Integration and Development

We need our model to be flexible according to the varying input data it receives. On top of that, model monitoring and performance evaluations are essential and responsible for ensuring that the models are working properly. However, doing this manually is tedious. This is where CI/CD(Continuous Integration and Development) can be useful.

CI/CD falls into the realm of DevOps but since we are building a machine learning model, it is called MLOps. The premise of MLOps is to enable developers to collaborate and develop AI models at a much faster pace. Developers can deploy, scale, monitor and retain their models at an increased pace.

Our project can take advantage of MLOps Level 1 where the goal is to automate ML pipeline solely to process continuous model training. By doing so, the model would provide predictions continually and automate the existing model in production using new data. This allows us to easily integrate the model to take online data, automate the creation, testing and deployment of the pipelines such that users can get their movie prediction immediately. [1]

The diagram below helps visualize the sequence of actions from source code to deployment in Level 1. [2]



2.3 Version Control

For version control, we have decided to use a GitLab repository to house all our code and supplementary files.

2.4 Timeline

Initially, we knew we had to start by collecting data for our project. Although using an existing dataset would have been much more accessible, we must account for real-time reviews. Thus, for our project's first stage, we decided to design a way to scrape IMDb effectively. Ultimately we split up the task of looking into different python libraries to decide which would be used to scrape reviews. Our first checkpoint of the project was completed once we had read through the selenium library and used it to scrape some sample data from IMDb. The next step would be to clean the data using NLP preprocessing.

2.5 Roles And Contributions

Data Collection Engineer: Grant Doan

web scrape data from the IMDb site, hyperparameter tuning, test model performance

MLOPS Lead/ML Engineer: Prayash Joshi

manage workflow, create automated pipelines, test model performance

Data Processing Engineer: Reagan Orth

data preprocessing, pipeline to Pytorch, model architecture

Data Engineer, UI Developer: Ved Patel

web scrape data from the IMDb site, deep data cleaning, website

2.6 Milestones

- Milestone 1: Gather and clean dataset
- Milestone 2: Initial neural network architecture
- Milestone 3: LSTM implementation
- Milestone 4: GNN implementation
- Milestone 5: Mean user prediction rating within three points of real ratings
- Milestone 6: Mean user prediction rating within two points of real ratings
- Milestone 7: Project complete

2.7 Tools

Tools that will be used throughout the project include but are not limited to:

- Python
- PyTorch
- pandas
- matplotlib
- numpy
- scikit-learn

2.8 Possible Approaches

The approach we have in our mind right now is simple and consists of four main steps. The first step is to scrape all of our data, which we have already made significant progress on. To scrape the data, we will use the selenium library and load all of the reviews of a particular movie. Doing this will allow us to have real-time reviews that are sorted according to our liking. The next step is to clean up our acquired data, which can be done through NLP preprocessing. One of the purposes of this is to clean up punctuation or any minor errors in the text. Another reason we want to do this is to create a sparse matrix with the text data. The sparse matrix will allow us to use various analysis methods on the text, including sentiment analysis. We will also want to use the sparse matrix to remove words that do not hold significance, such as names. After the scraping and preprocessing stage, we will create a long short-term memory neural network to learn with our non-sequential data. The use of the LSTM neural network will ensure that there are no vanishing or exploding gradients. Finally, we will create our graph neural network, which will be used to model the relationships between users and movies.

2.9 Workflow Options

2.9.1 Gitlab + JupyterNotebook + SQL on Local Machines

This is the simplest approach. Gitlab offers version control and JupyterNotebook through anaconda allows us to install libraries and packages to our working environment as needed. The disadvantage is the inconsistent working environments of all the collaborators which could lead to pointless debugging.

2.9.2 Gitlab + docker + JupyterNotebook + SQL on Local Machines

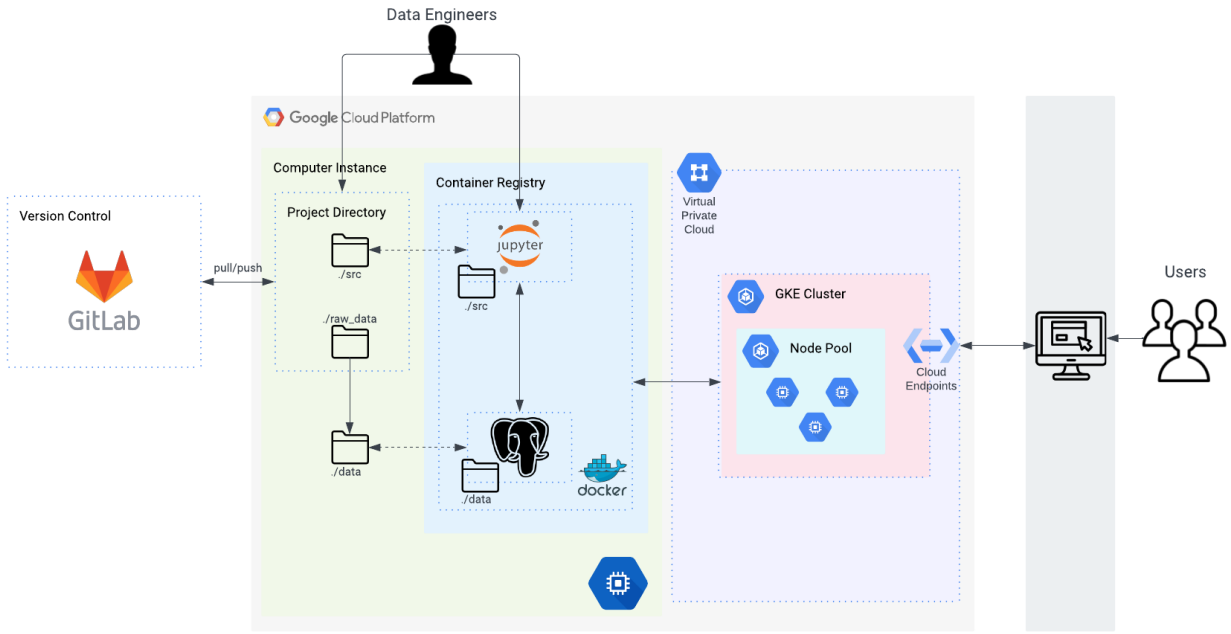
Docker Containers allows us to create a controlled working environment such that all the collaborators have the same libraries, packages and environment settings. The disadvantage of this model is that Docker can be CPU intensive.

2.9.3 Gitlab + docker + Jupyter Notebook + SQL + GCP Cloud Compute on Remote Server

Working on the cloud eliminates the worry of the local machine's bare-metal specs. All the processing is handled on a remote server. This allows for ease in running multiple concurrent Docker containers and leveraging as much processing power as we need from the Cloud provider. The disadvantage is that there is cost associated with maintaining a cloud infrastructure along with its services.

There are various other approaches that utilize the cloud that can make setting up the environment easy. However, the ease of use means higher monthly costs of maintaining the service. If our project moves to a cloud setup, we need to take the monthly costs into consideration.

2.10 Basic Cloud Architecture Diagram



References

[1] Satvik Garg, Pradyumn Pundir, Geetanjali Rathee, P.K. Gupta, Somya Garg, and Saransh Ahlawat. On continuous integration / continuous delivery for automated deployment of machine learning models using mlops. 2021.

[2] Google. Mlops: Continuous delivery and automation pipelines in machine learning nbsp;—nbsp; cloud architecture center nbsp;—nbsp; google cloud, Feb 2023.

[3] Ioannis Karamitsos, Saeed Albarhami, and Charalampos Apostolopoulos. Applying devops practices of continuous automation for machine learning. volume 11, page 363, 2020.

[4] Sandeep Kumar Rachamadugu, Jayanarayana Reddy Dwaram, and Kiran Rao Patike. Recommender system based on deep neural network and long short term memory. In *2021 International Conference on Computing, Networking, Telecommunications Engineering Sciences Applications (CoNTESA)*, pages 50–55, 2021.

[5] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. Graph neural networks in recommender systems: A survey, 2020.