YTViralPrediction-4641

Introduction/Background:

With the advent of ad-driven revenue models and creator-brand collaborations on social media platforms, content creation for many channels on YouTube has shifted from a fun side hobby to a main source of income. Though alluringly lucrative, social media is an inherently volatile industry, where consumers frequently jump between viral trends and favorite celebrities– leaving many creators struggling to find longevity and predictable success in their careers. To unveil the mysteries of audience behavior and the YouTube video-recommendation algorithm, great efforts have been made to discover relationships between configurable content features like title/description/thumbnail and key performance metrics like view-count/comment-count/click-through-rate (CTR). Medium led one notable initiative, applying machine learning techniques like NLP preprocessing, classification, and regression against video titles to predict CTR with roughly 70% accuracy (Paskalev, 2021). To draw our ML-driven conclusions, Team 41 has chosen the dataset Trending YouTube Video Statistics, containing video features such as title, description, publishing date, views, and likes (M. J, 2019). The known limitations of this dataset are its small size (3-24k records per English-speaking country) and its few user-configurable features (~6).

Problem Definition:

In a world where YouTube content creators are often fighting over the attention and loyalty of audiences, it is becoming increasingly essential to derive strategic insights from video engagement metrics (the number of views, likes, comments, etc.) to survive and thrive on the platform. Since the virality of videos can often feel random, with the YouTube algorithm viewed as a ‘black box’, our project aims to identify key video features that creators can leverage during the filming/production/editing process to maximize audience engagement.

Methods:

Preprocessing:

First, we aggregated all the English-speaking countries’ samples (United States (~6k), Canada (~24k), and Great Britain (~3k)) into one larger dataset. We originally intended on only using the U.S.’s, but opted for this new route instead to compensate for our weak features:samples ratio, increasing it from 10:6000 to 10:24000. This was done to avoid a high features:samples ratio that would more likely fit the noise in the data rather than the underlying patterns [1]. Afterward, we shuffled this stacked 3-country dataset to ensure the ordering/grouping of these countries would not influence unwanted patterns in the model during training. Next, we removed unnecessary features that do not contribute to the dependent variables we are measuring (e.g. video_id, ratings_disabled, etc), reducing the dimensionality from 16 to 11.

We then performed some feature engineering on the remaining raw column data, beginning by parsing the existing publish_time and trending_date columns into their respective dates and then taking their difference to produce a new feature to replace them: days_until_trending = trending_date - publish_date. This not only reduced the dimensionality of our dataset but more importantly transformed two columns that are not very insightful on their own into a more relevant metric for YouTube creators.

We also realized that there were duplicate videos within the CSV as the dataset was created by simply scraping the YouTube trending page every day. We joined these duplicates together and were able to engineer two new features as a result (days_until_trending and days_trending) as we felt they would provide meaningful labels as metrics of performance.

The last simple transformation step was just a matter of translating each video's integer category_id number into its respective text name (e.g. 5 → Sports). We did this integer → string conversion to prepare for our next step, encoding semantic meaning from all text features with vector embeddings. We chose to encode all the videos' string features with Hugging Face embedding [2]: title, description, channel title, category name, and channel tags (similar to 'bag-of-words', 'tags' is a list of the video's key terms and proper nouns). We decided vector embedding was a necessary step towards reducing the dimensionality of our dataset in its string features, which contained far too many characters to preserve the good feature:sample ratio we aimed for initially. Hugging Face was chosen mainly because it is free and well-documented.

Features (Note: Remove whichever feature is the label):

1. title_embedding (vector of floats)
2. description_embedding (vector of floats)
3. channel_embedding (vector of floats)
4. tags_embedding (vector of floats)
5. category_embedding (vector of floats)
6. days_until_trending (int)
7. publish_local_time_normalized (float)

Target Labels (rotate through these):

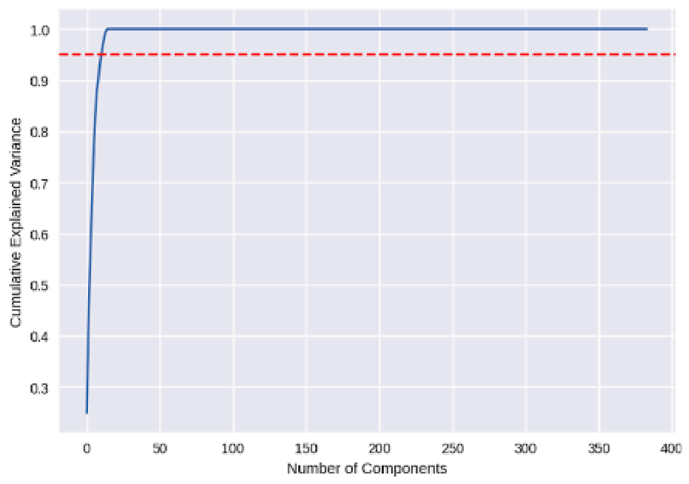
1. views (int)
2. comment_count (int)

3. likes (int)
4. dislikes (int)
5. like_dislike_ratio (float)
6. days_until_trending (int)
7. days_trending (int)

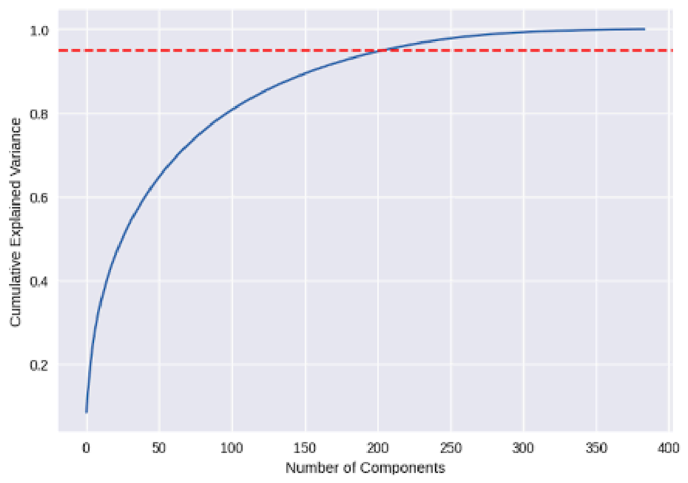
PCA

However, the hugging face model gave us embeddings that were each a vector of 384 values. If we were to flatten each of these embeddings and treat each value as a singular feature, we would end up with far too much dimensionality that could lead to overfitting. Initially, we were hardcoding to PCA with 300 values for each embedding, which brought our dimensions down from (33963, 1928) to (33963, 1508). However, each embedding had different variances, so instead, we implemented PCA with a threshold of 95% captured variance separately on each feature that was represented as an embedding.

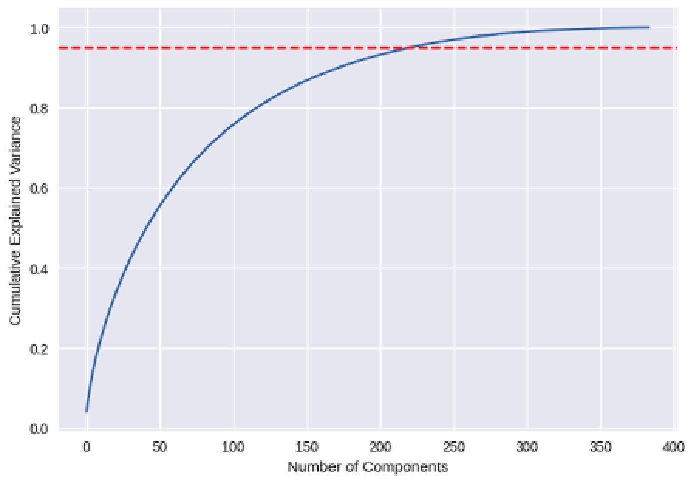
```
Ideal number of components to retain 95.0% variance for category_embedding: 11
```



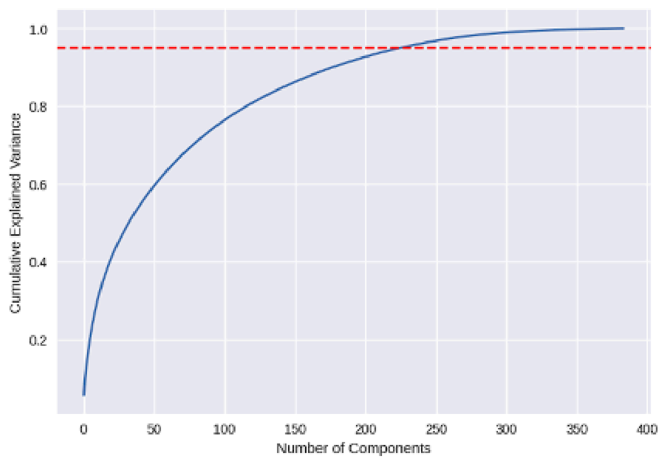
```
Ideal number of components to retain 95.0% variance for tags_embedding: 206
```



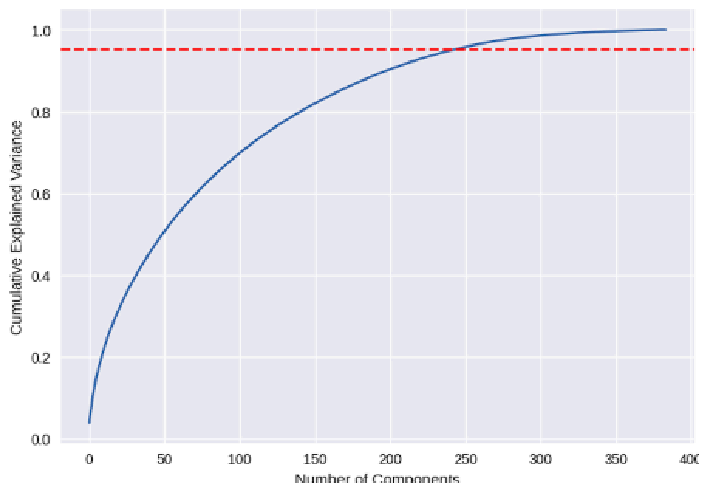
```
Ideal number of components to retain 95.0% variance for channel_embedding: 222
```



```
Ideal number of components to retain 95.0% variance for description_embedding: 226
```



```
Ideal number of components to retain 95.0% variance for title_embedding: 243
```

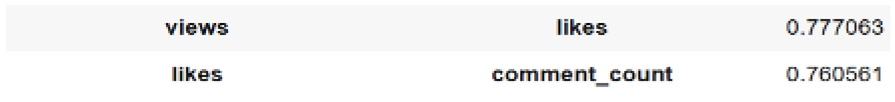


With this, we were able to get the ideal number of features to use for each text field: {‘title_embedding’: 243, ‘description_embedding’: 226, ‘channel_embedding’: 222, ‘tags_embedding’: 206, ‘category_embedding’: 11}. In the end, this reduced our data dimensions from (33963, 1508) to (33963, 916), a 1012 feature reduction from the original 1928.

Algorithms/Models

Linear Regression

We decided to start off with a simple model to be able to identify our data’s strengths/weaknesses/quirks more clearly before moving on to more complicated models. Given that we wanted to predict continuous values (views, likes, comments, etc.) from a few video features, linear regression was a natural choice. Initially, when we chose a particular label to evaluate, we had been using the remaining labels as features. This ended up causing a stronger correlation between some features than we initially realized (See Figure below; this impairs linear regression performance due to matrix singularity in the weights’ closed-form solution [3]). Thus, in the revision of our Linear Regression model, we stopped considering the extra labels as features. Additionally, we also found that changing the PCA from fixed at n=300 to dynamic n based on 95% variance capture improved performance.



views	likes	0.777063
likes	comment_count	0.760561

Correlations found between our features and labels in Model 1.0; Later fixed.

Ridge Regression

Since our dimensionality was still high (despite PCA), we knew that there was still a high chance of overfitting [5]. We wanted to preserve the aforementioned positives of linear regression, but also tackle this issue. We decided to do so with Ridge Regression, which is essentially a method where we limit how much we reduce our bias, but decrease overall variance in our model. This helps it combat overfitting. The alpha hyperparameter in ridge regression lets us tweak how much we want to limit bias minimization. We tried alpha values of 0.1, 1.0, 10, and 100 for each label ultimately finding that alpha=10 or alpha=100 performed the best depending on the label at hand.

Neural Network

We then decided to pivot to a model that could handle non-linear, more complex relationships between features. This is ideal when our model is reliant on features like text embeddings because neural networks are better able to handle ambiguity and semantics than other traditional models [4]. After some experimentation with layer numbers and depths, we found we got the best performance with 3 layers and a ReLU activation function. We chose ReLU due to the fact that neural networks tend to be more computationally expensive so choosing a computationally efficient activation function like ReLU would make it easier to experiment with different architectures. Also, we ensured that our layers went from larger depths to smaller depths. This is because we were starting off with a very large dimensionality, so decreasing depth size moving forward would allow us to remove many of the less-contributing nodes and work against overfitting [5]. Specifically, we went from (total_feature_count) → 64 nodes → 32 nodes → 4 (a subset of most relevant labels) output nodes.

Finally, since we were trying to solve a regression problem, we made our output layer have a linear activation function.

Results and Discussion:

Analysis of Algorithm/Model

Linear Regression

Channel Embedding Performance

Target	R^2	MSE	RMSE
Views	0.068	3.97e+13	6.30e+06
Comment Count	0.080	5.87e+08	2.42e+04
Likes	0.103	2.16e+10	1.47e+05
Dislikes	0.047	5.44e+08	2.33e+04
Like/Dislike Ratio	0.000	2.26e+06	1.50e+03
Days Until Trending	0.004	2.03e+04	1.43e+02
Days Trending	0.088	2.58e+01	5.08e+00

Description Embedding Performance

Target	R^2	MSE	RMSE
Views	0.069	3.96e+13	6.29e+06
Comment Count	0.057	6.02e+08	2.45e+04
Likes	0.106	2.16e+10	1.47e+05
Dislikes	0.029	5.54e+08	2.35e+04
Like/Dislike Ratio	0.000	2.26e+06	1.50e+03
Days Until Trending	0.015	2.01e+04	1.42e+02
Days Trending	0.121	2.48e+01	4.98e+00

Tags Embedding Performance

Target	R^2	MSE	RMSE
Views	0.078	3.92e+13	6.26e+06
Comment Count	0.074	5.91e+08	2.43e+04
Likes	0.120	2.12e+10	1.46e+05
Dislikes	0.031	5.53e+08	2.35e+04
Like/Dislike Ratio	0.000	2.26e+06	1.50e+03
Days Until Trending	0.015	2.01e+04	1.42e+02
Days Trending	0.101	2.54e+01	5.04e+00

Category Embedding Performance

Target	R^2	MSE	RMSE
Views	0.067	3.97e+13	6.30e+06
Comment Count	0.019	6.26e+08	2.50e+04
Likes	0.079	2.22e+10	1.49e+05
Dislikes	0.005	5.68e+08	2.38e+04
Like/Dislike Ratio	0.000	2.26e+06	1.50e+03
Days Until Trending	0.001	2.04e+04	1.43e+02
Days Trending	0.087	2.58e+01	5.08e+00

Title Embedding Performance

Target	R^2	MSE	RMSE
Views	0.077	3.93e+13	6.27e+06
Comment Count	0.047	6.08e+08	2.47e+04
Likes	0.108	2.15e+10	1.47e+05
Dislikes	0.017	5.61e+08	2.37e+04
Like/Dislike Ratio	0.000	2.26e+06	1.50e+03
Days Until Trending	0.013	2.01e+04	1.42e+02

Target	R²	MSE	RMSE
Days Trending	0.123	2.48e+01	4.98e+00

Combined Embeddings Performance

Target	R²	MSE	RMSE
Views	0.132	3.69e+13	6.08e+06
Comment Count	0.178	5.24e+08	2.29e+04
Likes	0.234	1.85e+10	1.36e+05
Dislikes	0.095	5.17e+08	2.27e+04
Like/Dislike Ratio	0.001	2.26e+06	1.50e+03
Days Until Trending	0.034	1.97e+04	1.40e+02
Days Trending	0.164	2.36e+01	4.86e+00

As seen in the above tables, the combined embedding outperforms the individual labels across all target labels, particularly for likes, comment count, and days trending. The individual embedding, tags performed well for predicting likes and views, while title embedding performed well on days trending. All the embedding struggled with predicting the like/dislike ratio – with near zero R-squared values across the board. Although the prediction of views improved with all embeddings, the RMSE remains quite high, suggesting that these metrics are difficult to predict given these features. The overall R-squared values of < 0.2 show that linear regression is not able to capture all the information needed to make the most accurate predictions.

Ridge Regression

Channel Embedding

Target	Best Alpha	R squared	Mean squared error	RMSE
views	10	0.068394	3.964697e+13	6.296584e+06
comment_count	10	0.075696	5.898047e+08	2.428589e+04
likes	10	0.101785	2.168469e+10	1.472572e+05
dislikes	100	0.027733	5.552000e+08	2.356268e+04
like_dislike_ratio	10	0.000337	2.258729e+06	1.502907e+03

Target	Best Alpha	R squared	Mean squared error	RMSE
days_until_trending	100	0.004577	2.030048e+04	1.424798e+02
days_trending	10	0.088705	2.575539e+01	5.074977e+00

Description Embedding

Target	Best Alpha	R squared	Mean squared error	RMSE
views	10	0.071408	3.951873e+13	6.286392e+06
comment_count	10	0.054492	6.033348e+08	2.456287e+04
likes	10	0.105669	2.159093e+10	1.469385e+05
dislikes	100	0.015938	5.619349e+08	2.370517e+04
like_dislike_ratio	10	0.000305	2.258801e+06	1.502931e+03
days_until_trending	100	0.014358	2.010101e+04	1.417780e+02
days_trending	10	0.121278	2.483479e+01	4.983452e+00

Tags Embedding

Target	Best Alpha	R squared	Mean squared error	RMSE
views	10.0	0.080327	3.913913e+13	6.256127e+06
comment_count	10.0	0.070739	5.929676e+08	2.435093e+04
likes	10.0	0.118172	2.128907e+10	1.459077e+05
dislikes	100.0	0.021361	5.588386e+08	2.363977e+04
like_dislike_ratio	1.0	0.000382	2.258627e+06	1.502873e+03
days_until_trending	100.0	0.012288	2.014322e+04	1.419268e+02
days_trending	10.0	0.103385	2.534049e+01	5.033934e+00

Category Embedding

Target	Best Alpha	R squared	Mean squared error	RMSE
views	100	0.067049	3.970423e+13	6.301129e+06

Target	Best Alpha	R squared	Mean squared error	RMSE
comment_count	10	0.018534	6.262802e+08	2.502559e+04
likes	10	0.078994	2.223492e+10	1.491138e+05
dislikes	100	0.004937	5.682170e+08	2.383730e+04
like_dislike_ratio	10	0.000199	2.259041e+06	1.503011e+03
days_until_trending	100	0.000938	2.037469e+04	1.427399e+02
days_trending	10	0.086820	2.580865e+01	5.080221e+00

Title Embedding

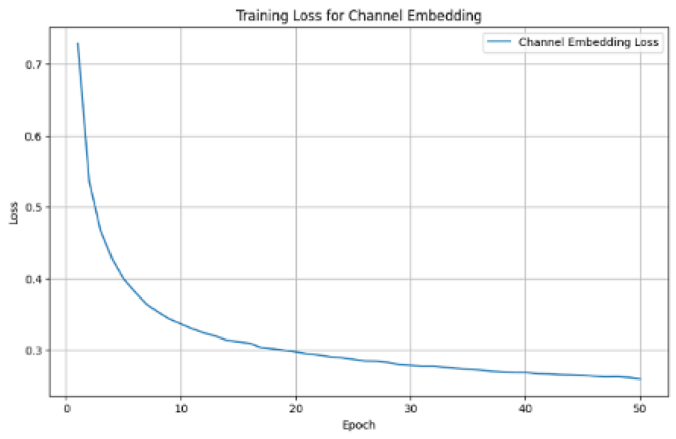
Target	Best Alpha	R squared	Mean squared error	RMSE
views	10	0.079627	3.916896e+13	6.258511e+06
comment_count	10	0.049263	6.066718e+08	2.463071e+04
likes	10	0.108744	2.151668e+10	1.466857e+05
dislikes	100	0.017868	5.608329e+08	2.368191e+04
like_dislike_ratio	10	0.000105	2.259254e+06	1.503081e+03
days_until_trending	100	0.011212	2.016517e+04	1.420041e+02
days_trending	10	0.125152	2.472530e+01	4.972454e+00

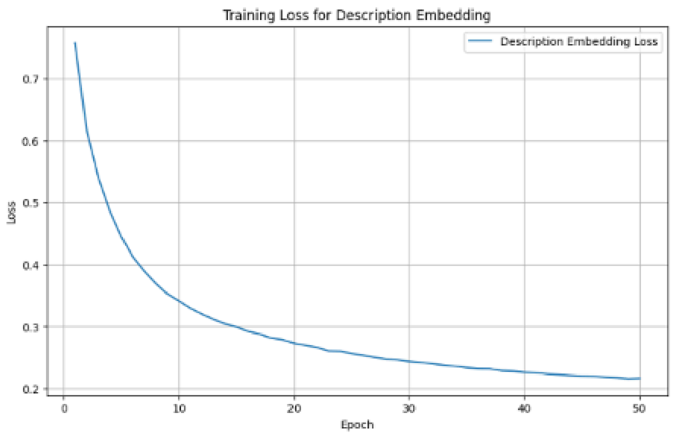
All Embeddings Combined

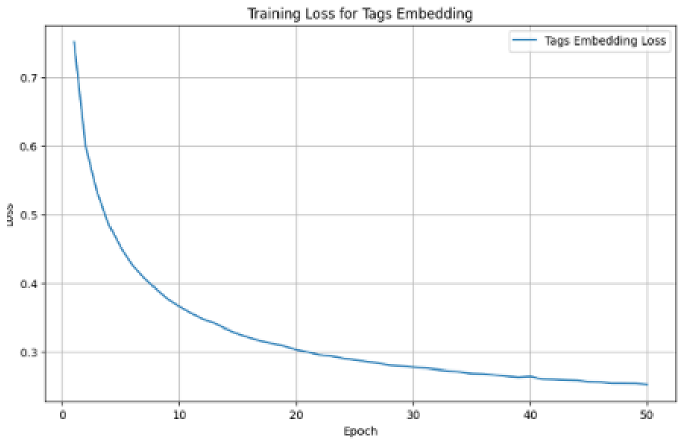
Target	Best Alpha	R squared	Mean squared error	RMSE
views	10	0.141922	3.651779e+13	6.042995e+06
comment_count	10	0.172692	5.279108e+08	2.297631e+04
likes	10	0.233360	1.850821e+10	1.360449e+05
dislikes	100	0.060467	5.365074e+08	2.316263e+04
like_dislike_ratio	10	0.000671	2.257975e+06	1.502656e+03
days_until_trending	100	0.034639	1.968739e+04	1.403118e+02
days_trending	100	0.174253	2.333759e+01	4.830900e+00

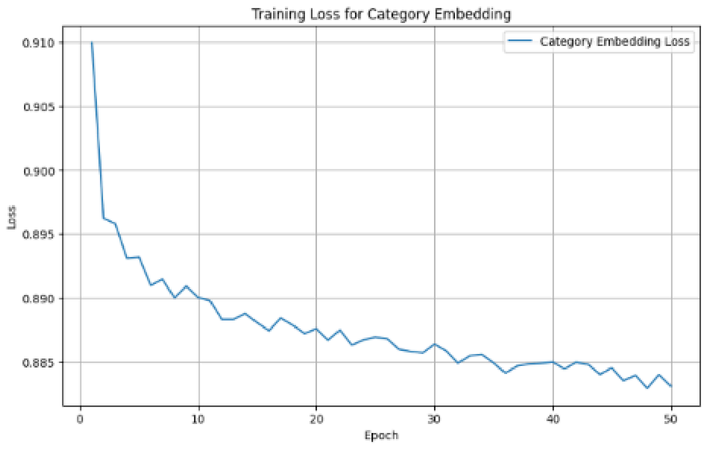
As seen by the results above, the trends and performance of ridge regression are similar to that of linear regression. We see that all embeddings significantly outperform the individual embeddings, with the highest R-squared values across almost all targets, especially for likes, days trending, and comment count. The individual embeddings were relatively weak with R-squared values typically below 0.12; however, the tag and title embedding performed slightly better than others when considering likes and days trending. Similar to the linear regression, all embeddings struggle with the like-to-dislike ratio. Optimal alpha values were 10 for most predictions while some targets like dislikes and days until trending were 100, indicating that moderate regularization is generally the most effective. There was a nearly doubling in the R-squared values for some labels when combined on the embedding, showing that all embedding provided the best results.

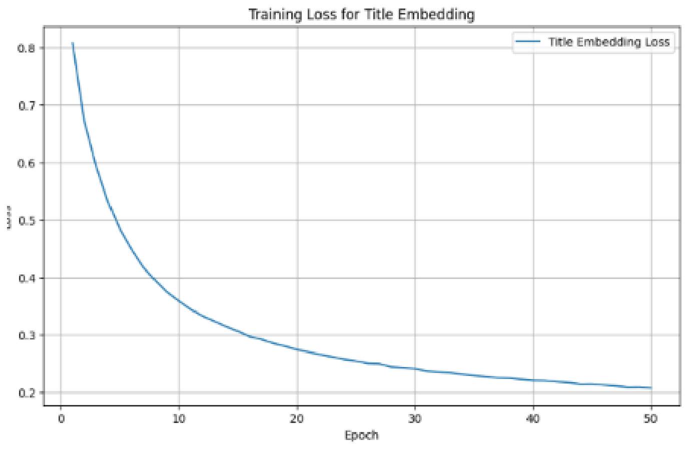
Neural Network

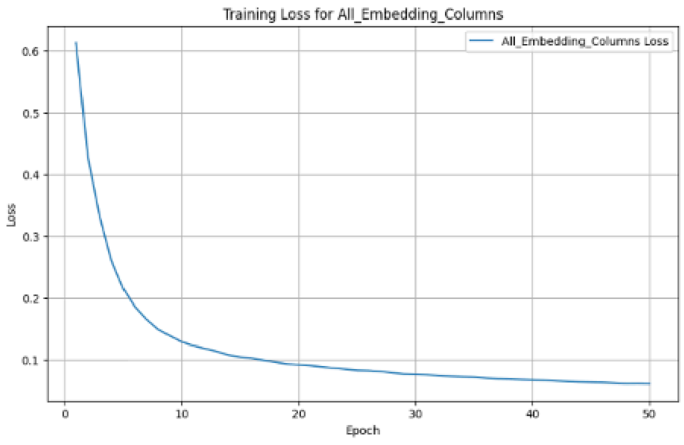












Embedding Performance Comparison

Loss Values Over Epochs

Epoch	Channel	Description	Tags	Category	Title	All Columns
1	0.725337	0.769729	0.759207	0.908922	0.802580	0.611795
2	0.537236	0.617585	0.605179	0.898464	0.666006	0.420861
3	0.468844	0.540347	0.537783	0.894931	0.589572	0.327360
4	0.431288	0.484518	0.490272	0.893906	0.526531	0.258442

Epoch	Channel	Description	Tags	Category	Title	All Columns
5	0.404324	0.441681	0.455713	0.892753	0.477572	0.213998
6	0.387105	0.411079	0.429100	0.891415	0.443780	0.184719
7	0.370168	0.388139	0.408645	0.890980	0.414970	0.161410
8	0.359573	0.368660	0.392612	0.890097	0.392145	0.146268
9	0.350840	0.353200	0.378481	0.889782	0.374466	0.135766
10	0.340405	0.338790	0.366423	0.888957	0.358474	0.127992

Performance Metrics

Embedding Type	R^2	MSE	RMSE
Channel	0.506384	0.489344	0.699531
Description	0.432021	0.636776	0.797982
Tags	0.417217	0.607034	0.779124
Category	0.342004	0.875763	0.935822
Title	0.329758	0.712877	0.844320
All Columns	0.365424	0.452360	0.672577

As seen in the data above, the Channel embedding emerged as the most effective approach with the highest R-squared value of 0.506384 and a relatively low RMSE of 0.699531. The All Columns embedding achieved the lowest final loss value of 0.127992 after 10 epochs, showing the most dramatic improvement from its initial loss of 0.611795. However, it ranked fourth in terms of R-squared value at 0.365424. Description and Tags embeddings performed moderately well, ranking second and third in R-squared values with 0.432021 and 0.417217 respectively. The Category embedding showed the poorest convergence, maintaining consistently high loss values throughout training (starting at 0.908922 and only reducing to 0.888957) and achieving the highest RMSE of 0.935822. This weak convergence was likely due to its embedding's extremely low dimensionality (only length 11 after PCA, compared to the 200s of the other embeddings). The Title embedding, despite showing good loss reduction during training, ended up with the lowest R-squared value of 0.329758, suggesting potential overfitting or limited predictive power of title features alone.

Comprison of Models

The three models—Linear Regression, Ridge Regression, and Neural Network—show the trade-offs between simplicity, interpretability, and performance in the context of predicting YouTube video performance. Each model offers unique advantages and limitations, driven by their underlying capabilities.

Linear Regression served as a straightforward baseline, offering simplicity and clear insights into feature-label relationships. However, it struggled with high-dimensional embeddings and multicollinearity, resulting in low R^2 values (e.g., $R^2 < 0.2$ for views and likes) and high RMSE. While unsuitable for final predictions, it identified preprocessing issues such as redundancies and low-variance features.

Ridge Regression enhanced predictive accuracy compared to Linear Regression by addressing multicollinearity and overfitting through regularization. With carefully tuned alpha values (ranging from 1 to 100 depending on the target label), Ridge Regression improved model stability and produced more balanced performance. For example, the model performed significantly better on metrics such as likes ($R^2 = 0.233$) and days trending, with RMSE reductions indicating better generalization to unseen data. Nevertheless, Ridge Regression's linear assumptions limited its ability to capture complex patterns in the data. While Ridge Regression outperformed Linear Regression, its improvements were incremental, demonstrating its usefulness primarily as an intermediate solution.

The Neural Network stood out as the most capable model, achieving superior performance by utilizing its ability to grasp non-linear relationships in the data. With a multi-layer architecture optimized for high-dimensional embeddings, it achieved the highest R^2 values across most labels, such as views (R^2=0.506) and likes, while also producing consistently lower RMSE values. This performance shows its strength in capturing patterns missed by linear models, but these gains came at significant computational costs. Training the Neural Network required careful hyperparameter tuning (e.g., learning rates, batch sizes) and extensive preprocessing to manage the high-dimensional data. Limitations in computational resources prevented more extensive cross-validation and experimentation with deeper architectures, which might have further improved results. Additionally, the reduced interpretability of Neural Networks presents challenges for understanding feature importance, particularly in embedding-based predictions. Despite these drawbacks, the Neural Network proved to be the most effective model for uncovering complex relationships in the dataset.

While Linear and Ridge Regression offer simplicity and efficiency, they fall short in capturing the complex relationships intrinsic to YouTube video performance data. The Neural Network, despite its resource intensity, delivers the most accurate predictions, making it ideal for scenarios where predictive power outweighs computational costs. Ridge Regression, on the other hand, provides a practical middle ground, offering improved performance over Linear Regression with less computational demand than Neural Networks. These models collectively underscore the importance of aligning model choice with project goals, available resources, and the complexity of the problem at hand.

Next Steps

1. Expand the Dataset:
    - Find or build a larger dataset with significantly more samples to improve model training and reduce overfitting risks.
    - Explore additional data sources, such as scraping YouTube API or aggregating public datasets.
2. Enhance Computational Resources:
    - Rerun the models on more powerful machines (e.g., using cloud services like AWS or Google Cloud for GPUs/TPUs).
    - Consider distributed training or parallelization techniques to accelerate the hyperparameter tuning and model evaluation process.
3. Optimize Neural Network Training:
    - Implement systematic cross-validation for neural network hyperparameters, such as learning rates, batch sizes, and architectures.
    - Test deeper and more complex architectures to potentially improve performance while monitoring overfitting.
4. Improve Embedding Interpretability:
    - Decode the “most important” embeddings by analyzing feature contributions to predictions.
    - Conduct targeted feature engineering based on insights to enhance input embeddings.
5. Broaden Target Labels:
    - Add more predictive targets (e.g., subscriber growth, watch time) that are valuable for YouTube performance analysis.

References:

[1] A. Vabalas, E. Gowen, E. Poliakoff, and A. J. Casson, “Machine learning algorithm validation with a limited sample size,” PloS one, https://pmc.ncbi.nlm.nih.gov/articles/PMC6837442/ (accessed Nov. 10, 2024).

[2] “Getting started with embeddings,” Hugging Face – The AI community building the future., https://huggingface.co/blog/getting-started-with-embeddings (accessed Nov. 10, 2024).

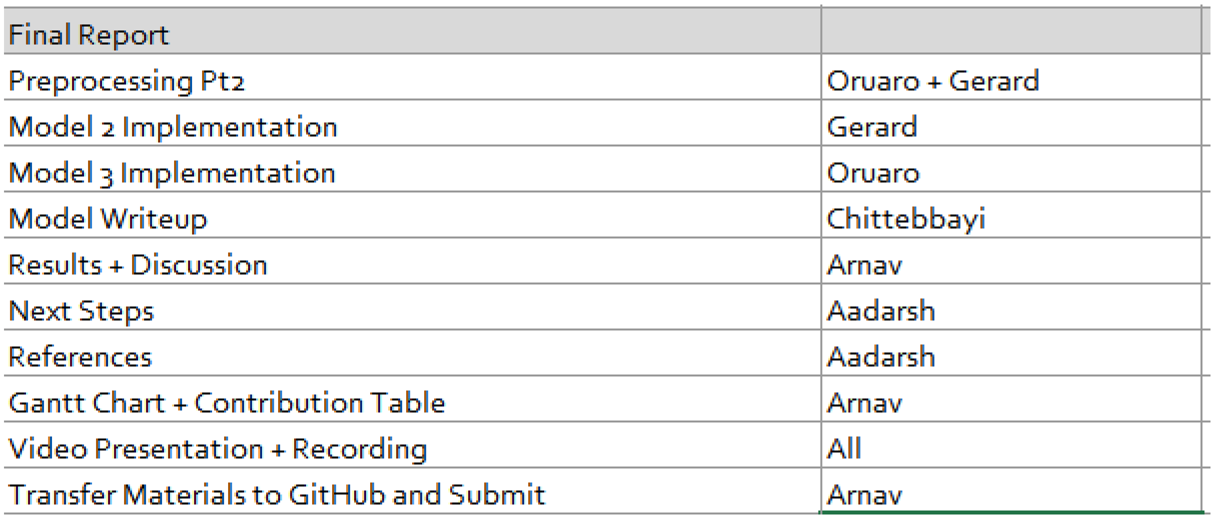
[3] K. P. Vatcheva, M. Lee, J. B. McCormick, and M. H. Rahbar, “Multicollinearity in regression analyses conducted in epidemiologic studies,” Epidemiology (Sunnyvale, Calif.), https://pmc.ncbi.nlm.nih.gov/articles/PMC4888898/ (accessed Nov. 10, 2024).

[4] L. Arras, F. Horn, G. Montavon, K. Muller, W. Samek, “What is relevant in a text document?: An interpretable machine learning approach,” PloS one,

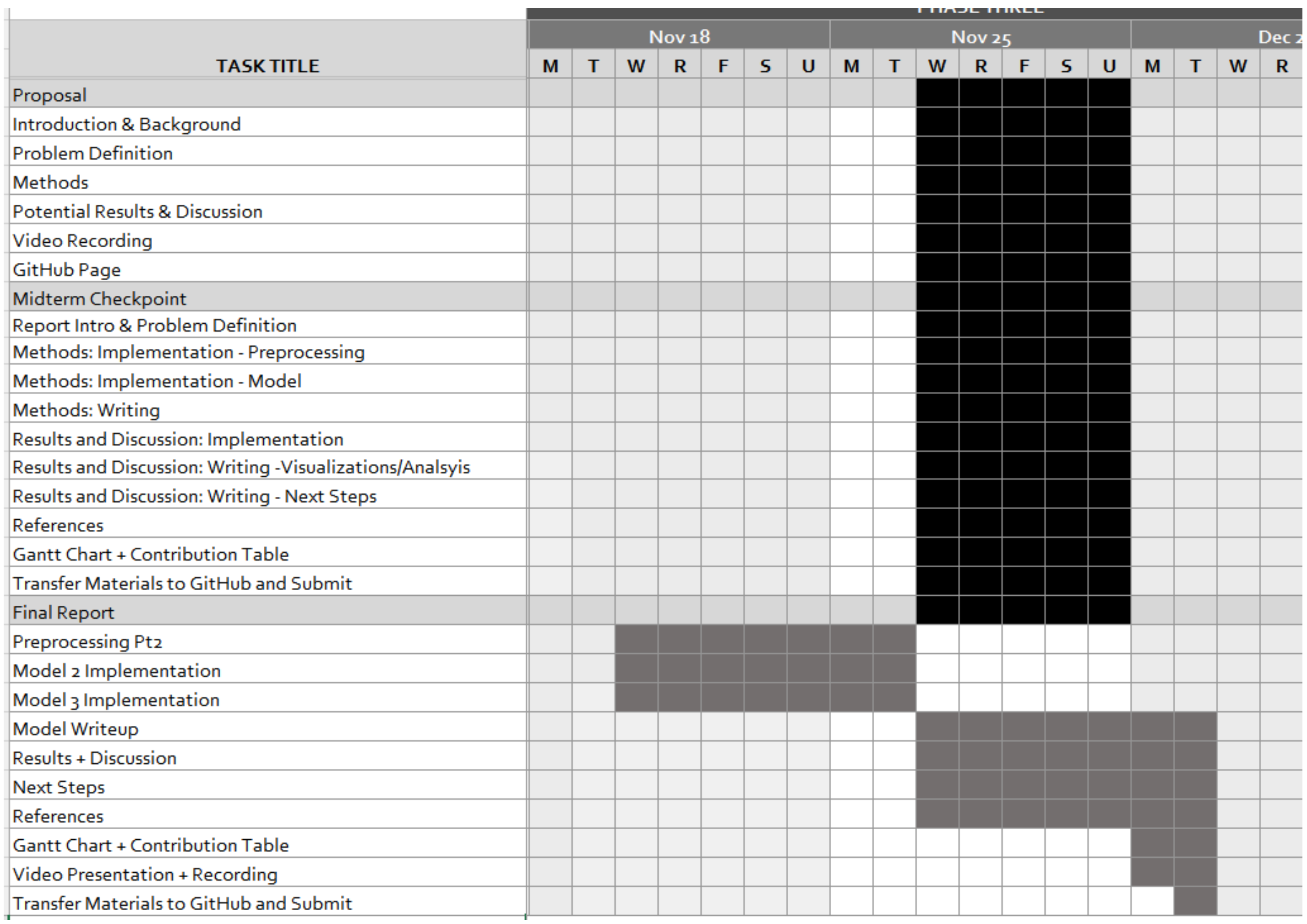
https://pmc.ncbi.nlm.nih.gov/articles/PMC5553725/ (accessed Nov. 20, 2024).

[5] X. Ying, “An Overview of Overfitting and its Solutions,” CISAT, https://iopscience.iop.org/article/10.1088/1742-6596/1168/2/022022/pdf (accessed Nov. 20, 2024).

Gantt Chart + Contribution Table



Final Report	
Preprocessing Pt2	Oruaro + Gerard
Model 2 Implementation	Gerard
Model 3 Implementation	Oruaro
Model Writeup	Chittebbayi
Results + Discussion	Arnav
Next Steps	Aadarsh
References	Aadarsh
Gantt Chart + Contribution Table	Arnav
Video Presentation + Recording	All
Transfer Materials to GitHub and Submit	Arnav



	PHASE THREE																	
	Nov 18							Nov 25							Dec			
TASK TITLE	M	T	W	R	F	S	U	M	T	W	R	F	S	U	M	T	W	R
Proposal										■	■	■	■	■				
Introduction & Background										■	■	■	■	■				
Problem Definition										■	■	■	■	■				
Methods										■	■	■	■	■				
Potential Results & Discussion										■	■	■	■	■				
Video Recording										■	■	■	■	■				
GitHub Page										■	■	■	■	■				
Midterm Checkpoint										■	■	■	■	■				
Report Intro & Problem Definition										■	■	■	■	■				
Methods: Implementation - Preprocessing										■	■	■	■	■				
Methods: Implementation - Model										■	■	■	■	■				
Methods: Writing										■	■	■	■	■				
Results and Discussion: Implementation										■	■	■	■	■				
Results and Discussion: Writing -Visualizations/Analsyis										■	■	■	■	■				
Results and Discussion: Writing - Next Steps										■	■	■	■	■				
References										■	■	■	■	■				
Gantt Chart + Contribution Table										■	■	■	■	■				
Transfer Materials to GitHub and Submit										■	■	■	■	■				
Final Report										■	■	■	■	■				
Preprocessing Pt2			▨	▨	▨	▨	▨	▨	▨									
Model 2 Implementation			▨	▨	▨	▨	▨	▨	▨									
Model 3 Implementation			▨	▨	▨	▨	▨	▨	▨									
Model Writeup										▨	▨	▨	▨	▨	▨	▨		
Results + Discussion										▨	▨	▨	▨	▨	▨	▨		
Next Steps										▨	▨	▨	▨	▨	▨	▨		
References										▨	▨	▨	▨	▨	▨	▨		
Gantt Chart + Contribution Table															▨	▨		
Video Presentation + Recording															▨	▨		
Transfer Materials to GitHub and Submit																▨		

Note: The full Gantt Chart can be seen on our github – It is `GanttChart.xlsx`