Intro. to Machine Learning
Final Project

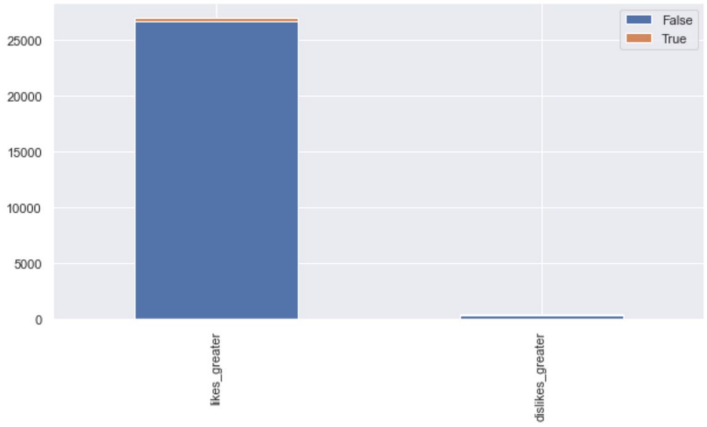
Raina Kim & John Cunniff

Introduction

Youtube is the most used video platform that contains tons of trending video lists. This dataset has data for these trending videos, and we analyzed them to get an idea of the correlation between these videos. Based on our analysis, our goal was to predict a chosen target (number of likes, category ids, and published in the morning or not) with corresponding features by using several different approaches.

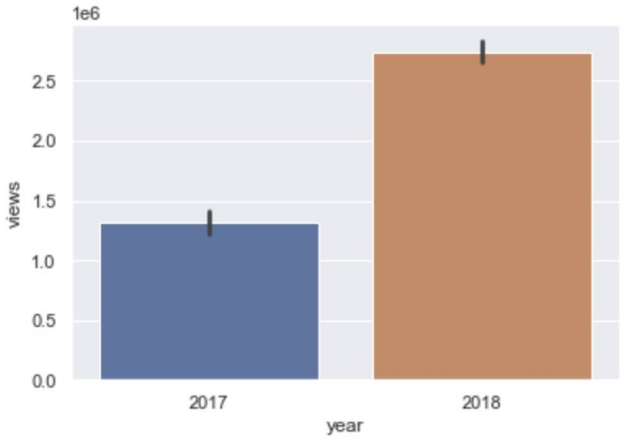
Unsupervised Analysis

- Association
    - How certain features of a data sample correlate with other features
    - We tried to get insights of how comments disabled option gets affected by whether the number of likes is greater or not. So does rating disabled.



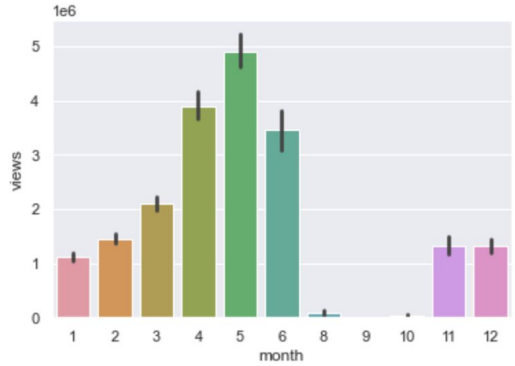
As we can see above, we concluded that we could not actually get a useful correlation between the number of likes and dislikes because only a few have greater dislikes.

- Feature Engineering
    - We derived features of published year, month, and likes from published time and removed unnecessary data, which is published earlier than 2017.
    - Then we visualized the number of views and likes with the published year.

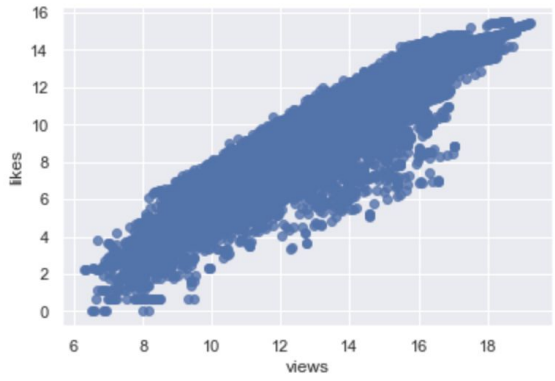


The overall shape of bar chart is almost the same for both views and likes since much more dataset were collected in 2018 than 2017.

- Number of views vs. months



- Finally, we got the shape of a linear model. By having the points lining up roughly like a thick but straight line, as seen below, we could get a hint of this relationship to be a linear regression.



Supervised Analysis

Linear regression

Overview:

Linear regression was pretty straightforward compared to other learning models. Only numeric values were used, so natural language preprocessing was not necessary at this point. Since we get the basic insights of the relationship between views and likes explicitly through the scatter plot, the process of building a linear model was not complicated at all.

What we wanted to predict was the trend of the number of likes based on the number of views. As we could see above, those features have linear relationships.

Preprocessing:

The only thing we had to keep in mind was to scale the numeric values that have a huge range in values, so we transformed the features to fit in the model properly.

Model Design:

We used built in linear regression from the sklearn linear model. The implementation itself is pretty simple since all are done by sklearn. We just had to pass in the scaled parameters.

Even the scatter plot explicitly indicates this relationship to be linear, but to prevent any overfitting, we added L2 regularization, ridge regression, on it. As predicted, there was no dramatic change at all in accuracy. It already had a good accuracy without regularization.

Problems faced:

There were no such difficulties for building linear models and its prediction also turned out to be straightforward.

Results:

The result table for linear regression:

	ridge	alpha	train accuracy	test accuracy
0	False	NaN	0.888640	0.879067
1	True	0.01	0.888640	0.879043
2	True	1.00	0.885195	0.873723

The ridiculous outcome was that the accuracy without the ridge regression was better than the one with the regularization. It indicates that the regularization is not needed for this case.

SVM

Overview:

The main point for svm modeling was feature transformation. In order to deal with NaN data, we could transform them to numeric values. The simplest thing would be to convert boolean value to 0 or 1. Also, we added some features that could be derived from publish time and made the datetime field to become classifiers.

What we wanted to predict was whether a certain video is published in the morning or not. We enriched the data so that we could set the derived feature as a target.

Preprocessing:

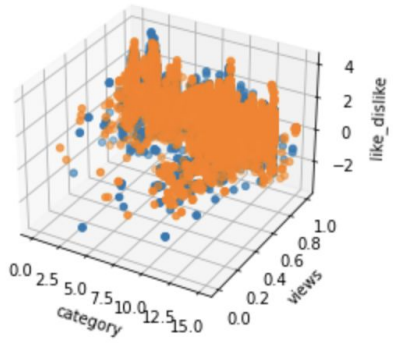
We tried to remove unnecessary values in between the text data and derive some features that seem to be useful when designing the model. We could merge the two separate data into a single column. Let's say there are 93 likes and 7 dislikes for a certain video. This data could be represented as a percentage of likes out of the overall reactions. We added a "morning_video" column which contains a boolean value and is derived from the publish hour. If the publish hour is less than 12, then it'll be considered to be morning and otherwise for any values greater than 12.

We also cleaned up the raw text data by dropping stop words and removing spaces and unnecessary vertical bars for any future uses.

Model Design:

We also used sklearn svm model selection package, GridsearchCV, for modeling, but before building the actual model, we first plotted our chosen features, category, like_dislike, and views with distinguishable colored dots, representing the morning_video values. We could see how

those data are related and distributed so that we could get an insight where and which type of hyperplane should be placed as a classifier to separate the data.

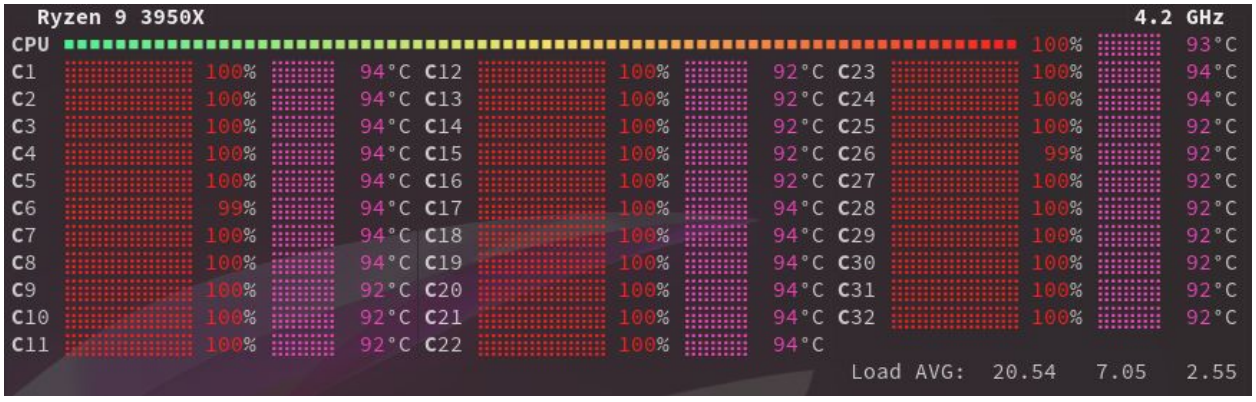


Based on our visualization, we chose to try a polynomial and rbf kernel with the gamma values of 0.01 and 0.001. The training time was reasonable.

Problem faced:

The results yielded were not promising. Given the proportion of videos that were published in the morning were ~0.311 (for the US region) our final svm models were no better than a 50/50 guess.

Some of the models took a very long time to train. Leaving a linear SVM training for over 24 hours yielded no results. We experienced laptops overheating, and desktops hard stopping due to overheating. When running the GridSearchCV, the CPU we worked with regularly exceeded 100°C. There was not a clear way to optimize the training by using a GPU. This is what the 32 core 3950X looked like when the GridSearch was running.



Results:

The result table for svm is shown below:

	param_C	param_kernel	mean_test_score
0	1	poly	0.691017
1	1	rbf	0.691017
2	1	poly	0.691017
3	1	rbf	0.691017
4	10	poly	0.691017
5	10	rbf	0.691017
6	10	poly	0.691017
7	10	rbf	0.691017

No matter how big the C parameter is and which type the kernel is, we roughly got the accuracy of around 69%, which is reasonable.

Convolutional neural network

Overview:

The more interesting features from the dataset were the text ones. We have the title and tags of all the trending videos. For our final model, we used a technique from natural language processing called embedding to feed text features to a CNN. The embedding process is quite simple. Simply convert strings of words into vectors of integers using a mapping of unique words to integers. This process is a dimensionality reduction that makes this type of neural network possible.

What we wanted to predict was what category of video a specific row was based on one of the text features. For example, given the title say “The Beatles - Come Together” with what amount of certainty can we say it is a music video or vlog. Therefore our model is a multi-classifier. We feed in some text, and out comes a vector of values between 0 and 1 showing its weight for each defined label or grouping. Our labels were each unique category of videos.

One of the more interesting things we were able to do is not only interpret english language videos, but some others as well. We were also able to create promising models for German and French language videos. Neither of us speak French or German, so this was very fun.

Our results were surprisingly strong.

Preprocessing:

An important part of the embedding process is that of tokenizing, then converting to integer vectors for each row of text in the input. In the past this consisted of two steps of preprocessing. The first step tokenization broke a given string of text into individual words. In some instances, we also drop what are called stop words from strings in this step. Stop words are words like “the” and “a”. By dropping these words from being considered, we choose to focus on the words that are more representative of the dataset. The second step converted the vectors of text into vectors of integers using the vocabulary mapping. This process is rather annoying to implement as there are a few places to make easy mistakes and generally requires using a library outside of keras. Thankfully, there is a new experimental preprocessing layer in keras called TextVectorization. This layer handles both those steps in the background for us. It also allows us to feed text directly into the model without having to convert it beforehand.

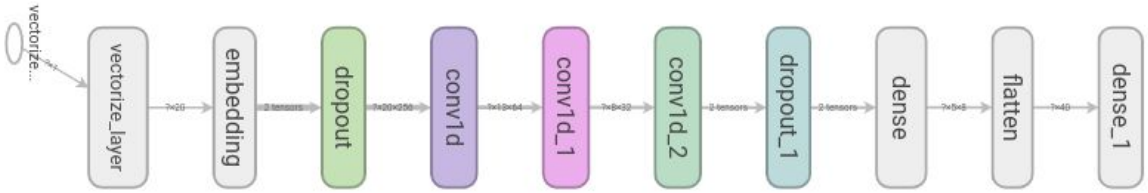
For the category of each video, we have an integer that represents the group it fits into. To convert this category integer to something that keras can more easily interpret, we pass it though to_categorical.

Model Design:

The model is a Sequential keras neural network. The top layers are the TextVectorization and Embedding layers. These handle the text processing portion of the model. The next layers are a mix of Conv1D, Dropout and Dense layers. The reason for the Dropout layers is to help prevent overfitting.

When training, we are using K fold cross validation to avoid overfit. One of the variable parameters we used was the number of folds. The other variable parameter was epochs per split. Using these variable parameters, we generated and trained models for

The final model as visualized through tensorboard is as such:



Problems faced:

The main problem we faced working with this type of model was both time and heat. When using GPU acceleration, the K80 we were running our training on repeatedly overheated. One

full training run where we create, train and test many models on the different parameters took several hours.

We would have liked to have more variation in the parameters, but we were limited by time and hardware constraints. Some of the parameters we would have liked to have variable would be layer activations, and weights per layer.

Results:

The results table for our CNN:

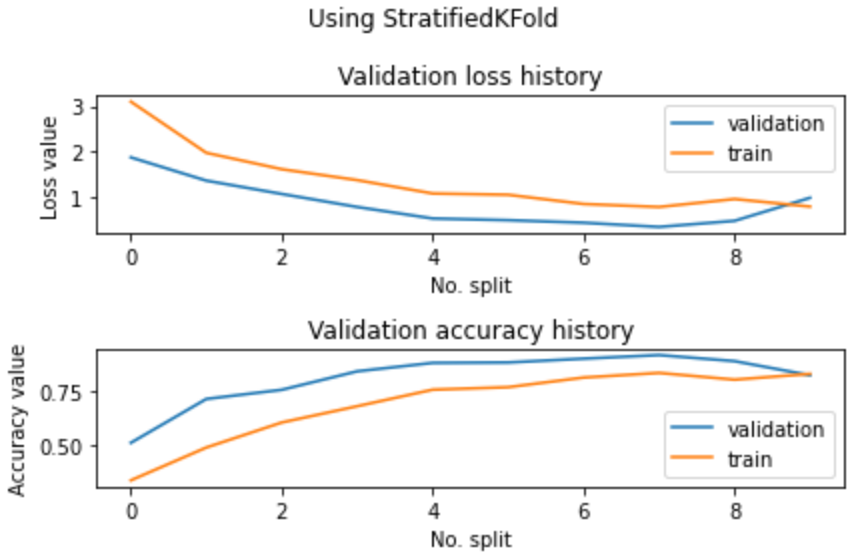
region	language	feature	epochs	kfolds	train time (s)	accuracy
US	english	title	1	5	17.39748836	0.5881109238
US	english	title	1	6	19.61044216	0.8610655069
US	english	title	1	7	23.02665973	0.9037256241
US	english	title	1	8	26.05184054	0.822124958
US	english	title	1	9	30.1233604	0.9254533052
US	english	title	1	10	32.18523359	0.9422885776
US	english	title	2	5	28.99495029	0.8697923422
US	english	title	2	6	35.55893397	0.9443366528
US	english	title	2	7	42.91902614	0.9616991878
US	english	title	2	8	49.04309773	0.9715479612
US	english	title	2	9	55.8649776	0.949182868
US	english	title	2	10	62.45204377	0.9298507571
US	english	title	3	5	43.13507605	0.848526299
US	english	title	3	6	53.41229796	0.9555290341
US	english	title	3	7	63.17824888	0.932277143
US	english	title	3	8	71.15458131	0.759251892

						6
US	english	title	3	9	84.26539254	0.8518021107
US	english	title	3	10	93.39432478	0.9554726481
US	english	tags	1	5	17.06440639	0.8491480947
US	english	tags	1	6	20.77836943	0.8139083982
US	english	tags	1	7	28.4045279	0.8824860454
US	english	tags	1	8	27.42541957	0.9204138517
US	english	tags	1	9	31.84399629	0.9113498926
US	english	tags	1	10	34.72881746	0.9054726362
US	english	tags	2	5	30.70565057	0.9105832577
US	english	tags	2	6	38.24982142	0.9344874024
US	english	tags	2	7	45.66203928	0.8897980452
US	english	tags	2	8	53.54559898	0.9309589863
US	english	tags	2	9	60.60575366	0.9265726209
US	english	tags	2	10	67.25922775	0.9405472875
US	english	tags	3	5	46.01881957	0.9427931905
US	english	tags	3	6	56.41756201	0.9313535094
US	english	tags	3	7	67.56648088	0.9524721503
US	english	tags	3	8	79.38709164	0.9426979423
US	english	tags	3	9	89.48213959	0.9543317556
US	english	tags	3	10	99.80300212	0.936815917

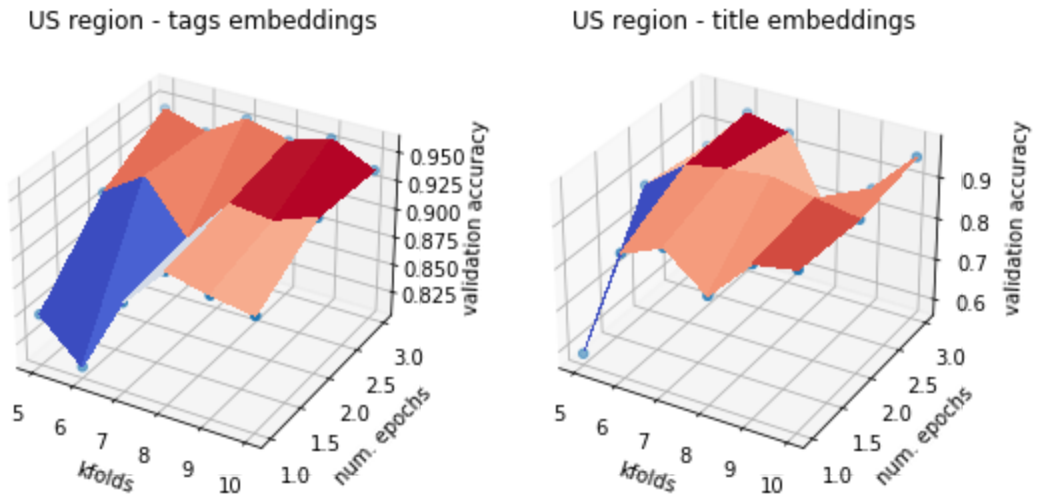
						5

In the interest of conciseness, the above table is only for the US region. The results for the France and Germany regions were similar.

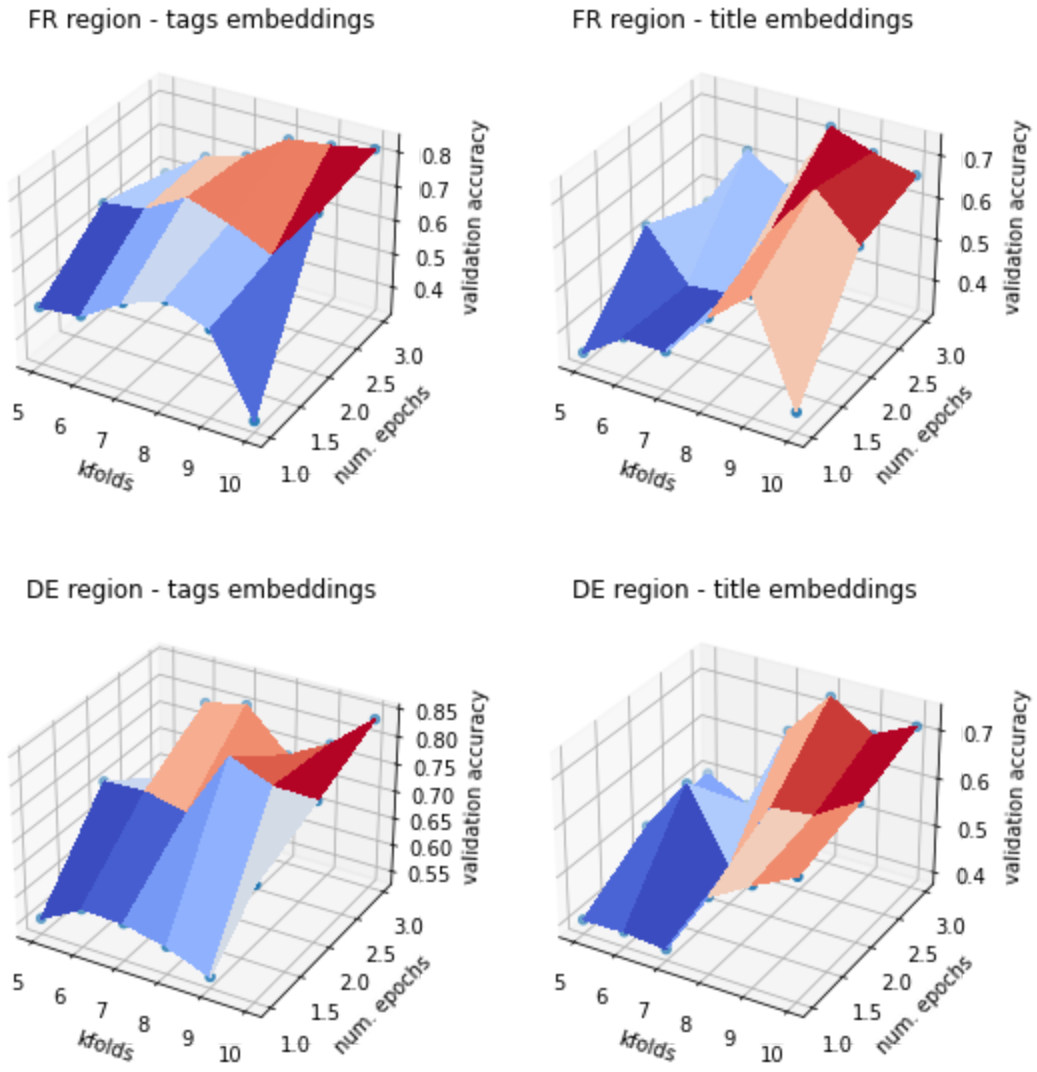
The following graph shows the training and validation loss and epoch over epoch iteration for a single model.



To visualize the accuracy in a more intuitive way, we can plot the numerical parameters and accuracy in 3 dimensional space, then apply a contour plot between the points. This gives us these very interesting looking graphs.



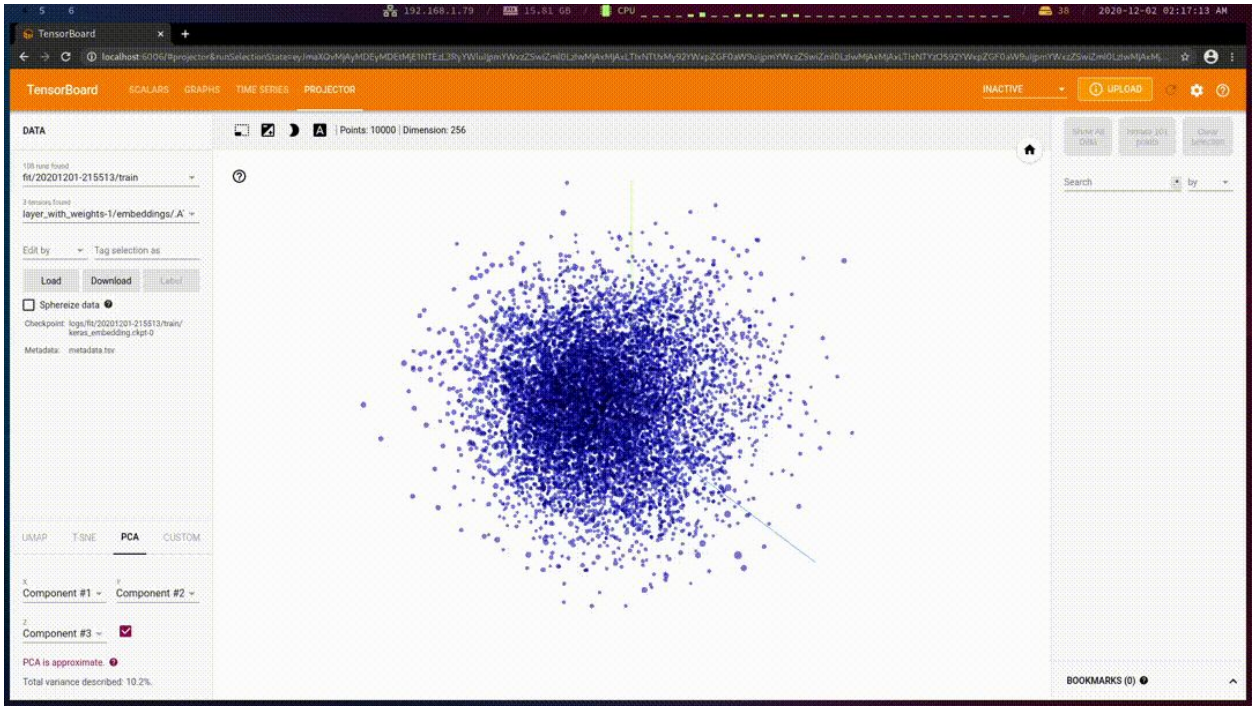
The left graph uses the video tags as the input, while the right the video title. Using this visualization we can spot the “sweet spot” much easier. It is where the point is highest, and where the contour is red.



The higher accuracy models attenuated towards when tags were used as input, kfolds were greater and number of epochs was greater.

Another visualization we can do to get a better understanding of the types of relationships the CNN is making is using a PCA projection. This type of projection brings the multi dimensional relationships between word embeddings down to a more bit sized format. We can create a PCA projection in tensorboard. The PCA projection is a good way to show the strength of relationships between words in the vocabulary of the model.

In the following gif, we show the words that were found to have a strong relationship to the word Brooklyn.


TensorBoard
SCALARS
GRAPHS
TIME SERIES
PROJECTOR
INACTIVE
UPLOAD
DATA
Points: 10000 | Dimension: 256
fit/20201201-215513/train
layer_with_weights-1/embeddings/.A
Edit by
Tag selection as
Load
Download
Label
Sphereize data
Checkpoint: logs/fit/20201201-215513/train/keras_embedding.ckpt-0
Metadata: metadata.tsv
UMAP
T-SNE
PCA
CUSTOM
Component #1
Component #2
Component #3
PCA is approximate.
Total variance described: 10.2%.
Search
by
BOOKMARKS (0)