CS109B - Project - Milestone 5
Spring 2017, Harvard

Team
- Angela Ambroz
- Keun-Hwi Lee
- Johanna Ramos
- Pranav Sidhwani

Overview

How do we decide which movies to watch? We might see posters or advertisements, read about it on a reviews website, or hear about it on social media. In each of these instances, we are latently absorbing information about the movie's likely genre. In all likelihood, we are employing heuristics that could be replicated by an algorithm or statistical model: for example, we expect that movies with the word "love" in the overview are more likely to be romantic comedies; we expect that dark, moody posters are likely to be horror movies.

For our final project in the Spring 2017 class of CS109B - Advanced Topics in Data Science, we employed a variety of methods to answer a seemingly simple question: can we train a model to accurately predict movie genres? What sort of data would be most predictive? Would a straightforward statistical model be as performative as a neural network?

In the end, we were able to predict movie genres at about 40% - better than random, but not as good as a human. (Though, to be fair, we did not check this by actually testing our own predictive accuracy!) We explored using natural language processing and predicting movie genres using the words in each movie's summary. We fed poster image data into a convolutional neural network. And we used movie meta-data, such as release date, cast, and crew, in a Support Vector Machine.

We will discuss the specific challenges that arose at each stage of the project process, as well as ideas for future exploration.

Data

We began with a sample of all movies from 1970-2016 (obtained from The Movie Database's (TMDb) API): this was over 200,000 movies. We then filtered only to movies that were (1) in English, (2) had an associated poster image, and (3) at least one genre. This reduced our dataset to over 51,000. We then randomly sampled 5,000 movies from this dataset of 51,000 in order to conduct our exploratory data analysis and initial models.

Regarding features, we used many that were readily available from TMDb's API: release date, revenue, a measure of popularity, runtime, and so on. For the poster images, we downloaded the smallest available size (color images with a width of 92 pixels and variable height).

Prediction problem

The first decision we had to make was whether to predict a single (or pair of) genre(s) per film (i.e. a multi-class problem), or whether to predict each film's unique set of genres (i.e. a multi-label problem). Movies had arbitrarily long sets of genres each: for example, movie X might be a [Comedy, Drama, Romance], while movie Y would be a [Horror], and movie Z a [Historical, Epic, Drama]. Certain genres were very highly represented (Drama), while others appeared infrequently (Sport). Furthermore, genres were frequently co-occurring, so that there was an element of clustering across genres.

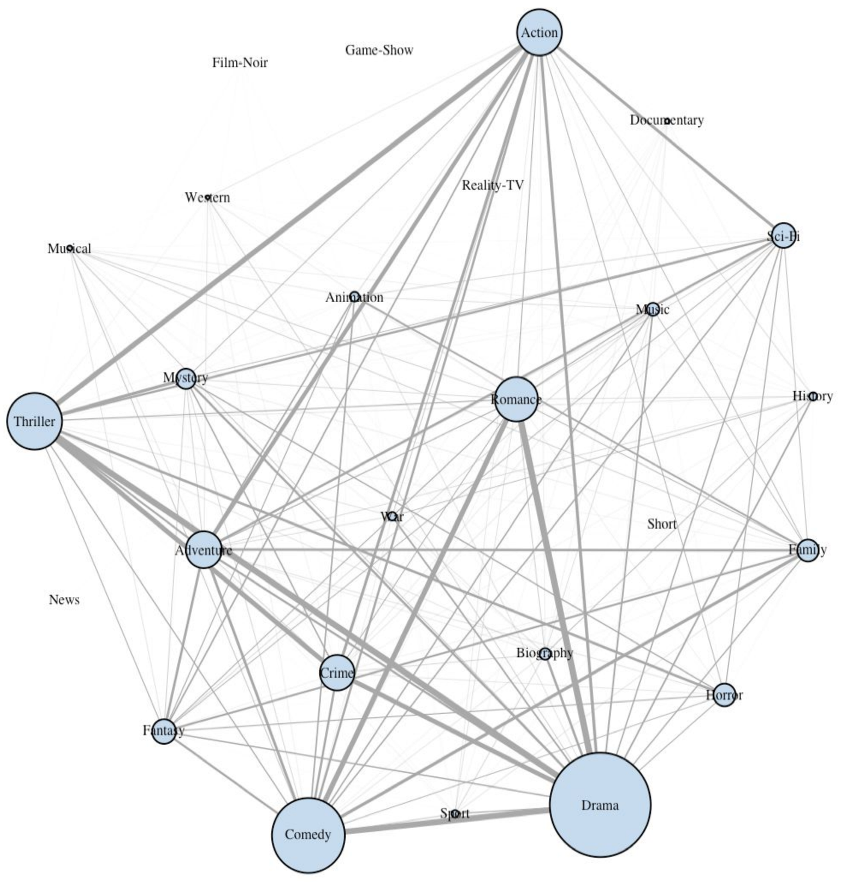


Figure 1: A network graph of our sample's movie genres

Our decision, then, was whether to predict movie genres as a multi-class or multi-label problem. There were pros and cons to both. For example, predicting single genres or bigrams of frequently co-occurring genres greatly reduced the complexity of our prediction problem. That would, we anticipated, make it easier to train a model and lead to a boost in accuracy. However, it was potentially over-simplifying, and infrequently occurring genres would be less likely to be represented (though we could correct for this with up-sampling the infrequent genres).

If we treated our prediction problem as multi-label, then we would be increasing the complexity of our problem - which would likely then require more data to obtain good accuracy. However, we would likely improve our real world applicability, and avoid over-simplifying reductivism (e.g. Romantic-Comedies would be reduced to simply Comedies, ignoring important differences in the genres).

Models

We tested a variety of models:

- ***Support Vector Machines***
  - We tested a Support Vector Machine (SVM) with a radial basis function. We tried including and not including class weights (to account for the class imbalances observed); and we tried using, for one set of models, only the text features (word-frequency vectors for the top 400 words from movies' summaries), and, for another set of models, the meta-information as well (a movie's budget, year of release, director, and so on). We predicted movie genre bigrams, as well as the multi-label problem. Below, we report on our multi-label prediction.
- ***Naive Bayes***
  - We constructed word frequency vectors and used a Dirichlet-multinomial "bag of words" naive Bayes model to predict movie genre bigrams. We tried this with and without class weights as well, and tried three prediction problems: popular bigrams (reported below), and multiple rows per genre per movie.
- ***Convolutional neural network (CNN)***
  - Our CNN had four convolutional layers, three pooling layers, and two dropout layers. We used the sample of 5,000 movies (reduced to 4,722 in data cleaning), a binary cross-entropy loss function (i.e. logistic function), rectified linear unit (ReLU) activation functions on all intermediate layers (excepting the output layer, whose activation function was sigmoid), and a stochastic gradient descent optimizer with a learning rate of 0.001. We ran the network for 10 epochs. We predicted multi-label movie genre sets.
- ***Pre-trained CNN***
  - We used a pre-trained convolutional neural network, Xception, which is a network pre-trained on the ImageNet dataset. We added four fully-connected layers on top of the existing architecture (which was already much deeper and more complex than our network - over 22 million parameters, compared to our 73,000).

Model	**Data**	**Features**	**Multi-class vs. Multi-label**	**Accuracy**
Support Vector Machines	5,000 sample ~2,000 test	Movie meta-information	Multi-class	25% (overall)

Support Vector Machines	5,000 sample ~2,000 test	Movie meta-information and summaries	Multi-label	9% (overall) 52% (precision) 40% (recall)
Naive Bayes	5,000 sample ~2,000 test	Movie summaries only	Multi-class	26% (overall)
Convolutional neural network (CNN)	5,000 sample ~2,000 test	Poster image data	Multi-label	44% (overall) 10% (precision) 50% (recall)
Pre-trained CNN	5,000 sample ~2,000 test	Poster image data	Multi-label	23% (overall) 10% (precision) 49% (recall)

Examining the precision and recall of our three multi-label prediction attempts (respectively, the SVM, CNN, and pre-trained CNN):

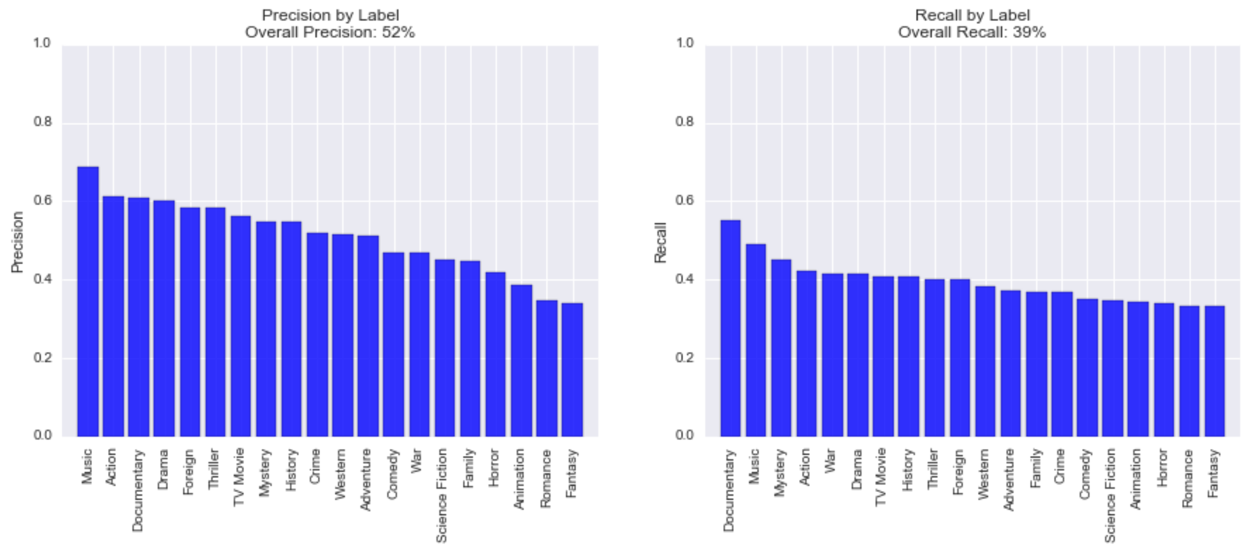


Figure 2: Precision and recall for the SVM

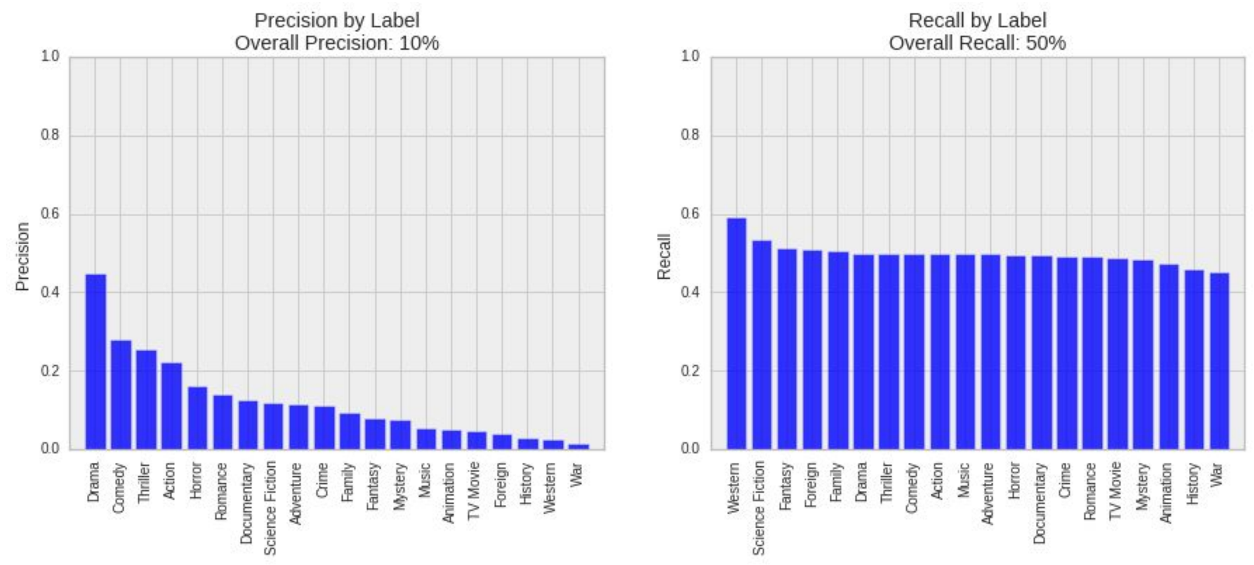


Figure 2: Precision and recall for our convolution neural network

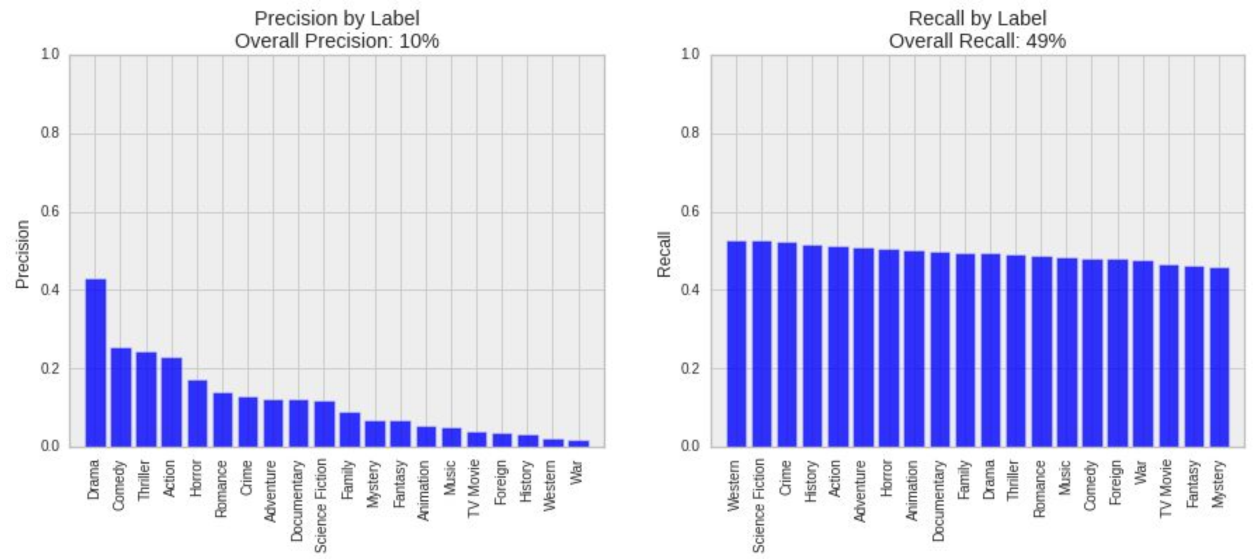
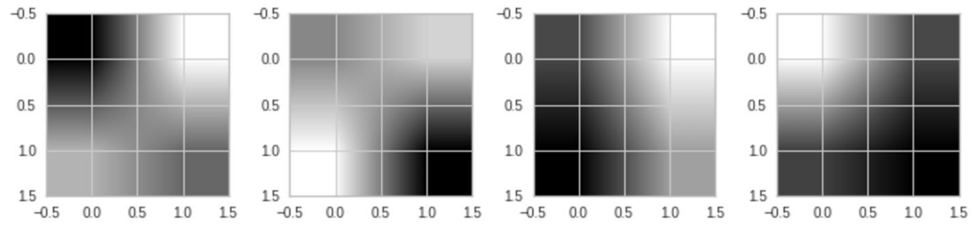


Figure 3: Precision and recall using a CNN pre-trained on the Xception architecture

Examining the first few filters of our CNN, we see that our neural network quickly and mostly picked up on corners:



Challenges
We faced several challenges during this project - challenges which, unfortunately, also exhibited interaction effects.

One challenge was the technological overhead of Amazon Web Services. We used p2.xlarge EC2 instances, and memory issues when loading image sizes larger than 92 pixels, and movie samples larger than our 5,000. We experimented training in batches, but encounter technical issues. File size also became an issue in Python's pandas library: for example, pandas returned a memory error when we tried to read the CSV of 50,000 movies.

A second challenge was class imbalances and the relative infrequency of certain genres. This would have been solvable if we could have obtained a much larger sample of data - in that case, we believe the neural net could have performed significantly better.

Conclusion
Interestingly, the CNN with poster image data alone did not outperform our SVM using text data and meta-information. This resonated with what we learned earlier in the course, regarding SVMs sometimes outperforming neural networks, especially for datasets smaller than some N. Given the technological overhead which running neural networks necessitated, and the SVM's ability to deliver significantly better precision for the same amount of recall, we would likely implement an SVM algorithm, if implementing this in a production environment.

Appendix and guide to code
All code for this project is available on GitHub: all milestone notebooks can be found here.

Specific notebooks are here:

- Support Vector Machines
  - **Multi-label (best performing model, as measured by precision and recall)**
  - Multi-class (predicting bigrams)
- Convolutional neural networks
  - Original model
  - Pre-trained model (Xception)
- Naive Bayes
  - Multi-class (predicting bigrams)

Our final dataset is available here:

- Training data
- Test data
- Zip to everything