Introduction

The purpose of my thesis project is to design an algorithm for taking a film script and systematically generating a shot list. On typical motion picture productions, creating a shot list is the collaboration of key members of the film crew, most typically the director and director of photography (often called the DP). Other artists might join as storyboards are created and decided upon. Deciding on shot lists is an artistic process, and the creative engineers use years of experience to decide what and when. As it could be argued that this process is done with intelligence, (although others may argue it's artistic instinct), I will create a program that uses artificial intelligence to complete this process.

What is a shot list? In its most basic form, a **shot list** is a mostly chronological list of shots that covers every moment in the script. Let's break that down further and ask, what is a shot? In filmmaking, a **shot** is a continuous piece of film with no breaks. It's likely called a shot, because it was shot with a camera. A shot can be **close up** on the subject, or far away. It can be static, or the camera can be moving in a variety of ways. The subjects can be framed in many ways. It can contain one or many subjects. Traditionally, if you wanted to switch between one shot and another, you literally had to cut the film with a blade of some kind, and then splice the pieces of film together. (A dying art with the digital era, but highly exhilarating). In editing a film, this switching from one shot to another is called a **cut**.

A traditional shot list will often have multiple shots covering an aspect of the script. For example, a protagonist giving a speech may be filmed with a close-up as well as a wide shot. Shots of his audience might also be filmed. The shot list might not be strictly in chronological order and multiple cameras / takes of the same scene can be filmed to have footage to cut between these shots. This allows for deciding after filming is complete where exactly to put the cuts and exactly which shot to use for each moment. However, once the film is completely shot and edited, there exists only one shot list in the sense that every moment is covered only once. The shot types are well defined, as well as when cuts happen.

The goal of this project is to produce shot lists of the latter kind, as if it represents a finished film. Still, like the traditional method, a physical script will be read in and appropriate shot types and cuts will be chosen. Since directors and cinematographers use their years of experience to

intuitively pick the shots, this program will also use their years of experience. This will be done in the form of training sets.

Here is the basic breakdown of how the project will work: Scripts will be read in using a parser. A parsed script can be used either as the basis for creating another element of the training set, or be fed into the shot list tool that uses previous training sets to come up with a shot list. Creating the training sets will be done with a liner tool that will take the parsed script and allows a user to manually add the training set information.

We now discuss the organization of the rest of this document. It will cover in more detail the four deliverables of CS297. First, the Parser, which in short parses a script into a form readable by the rest of the tools. The Liner is used to line scripts to make training sets. The third deliverable is the initial Training Set. Training sets in general will be discussed further. Finally, the Lister which implements the main functionality is explained.

The Parser

The first tool created for this project was the script Parser. The basic function of the Parser is to take a script and convert it into a serialized Java script object which can be exported as a JSON or used in the Liner and Lister tools. It works in a similar way to most parsers. It reads the text and tries to determine based on a set of rules what each thing is.

Hollywood film and TV scripts are supposed to follow certain conventions, although there are certainly many deviations. The Parser tool has been programmed to go by these conventions while reading through the script. Scriptwriting convention says for example that a new scene should start with “EXT.” or “INT.” The first is short for exterior, meaning outside, and the second is short for interior meaning indoors. The Parser has conventions for dialogue, and action blocks too, characters and certain objects.

The Parser goes through the script line by line with the primary purpose of determining what each line is. For example, is it a new scene? Is it a character speaking? Is it an action? Transition? The parser analyzes each line and uses the scriptwriting conventions to figure out what it is. Once it has categorized, it can add to the script data structure.

The script data structure contains some basic metadata about the movie as a whole, which can later contribute to the artificial intelligence analysis. It also contains a list of scenes. It contains a list of script objects and it has scene objects explained more below. Finally, it has a list which has a reference per line back to the scene object.

Each scene contains references in chronological order to all the scene objects contained in that scene. It also contains a list that references all of the script objects that appear in that scene. A scene object can be any of the above mentioned things, like a new scene marker. It can be a dialogue block. It can be an action block. It can be a transition. Even blank lines are recorded.

A script object is anything the parser has calculated to be an object worth keep tracking of. They usually appear in multiple scenes throughout the script. The most important script object is a character. Every dialogue block also references a character script object. Other types of script object include a prop which could be any inanimate object that characters interact with like a book or a gun. Objects might be a piece of scenery. The parser can detect script objects in two ways. First of all, if it finds a dialogue block, it automatically knows the character name. Scriptwriting conventions put the character's name at the top of a dialogue block. The parser can also find important objects because they've been put in action blocks with all uppercase letters. It is, for example, script writing convention to put a character's name in all uppercase the first time the character is introduced. It's also scriptwriting convention to make props and other important objects uppercase. Some actions, sounds, even camera movements are put in uppercase.

The parser is designed to not include duplicate script objects . So, regardless of how a character is discovered, whether by being introduced in a dialogue, or action block, it's only recorded once. Of course, if the name is spelled differently, or the character given different names, that's a different matter, but on the whole, that still works okay. The parser is also pretty good about throwing out extra characters such as commas and periods when checking to see if objects have already been added to the script object master list.

Other pieces of data collected include all the scenes each object appears in. Ultimately, all these pieces of data can be used as part of the artificial intelligence algorithm. Since every dialogue references the character, how many lines a character speaks can be extracted. Even after the parser completes once, it goes back over the script again, to make sure any references to an

object weren't missed. Not all scripts follow the convention of making characters uppercase in action blocks, so this is an important step.

The script object created by the parser is mainly for use with the Lister tool, in order to actually run the Naive Bayes (or other algorithm), but it also provides some information that the Liner tool needs to effectively run.

The Liner

The Liner tool's primary purpose is for creating training sets for the Lister tool. I call it the Liner tool because of the old technique of lining a script. This was basically the process of marking up a script with lines to denote shots. It is done both before production by directors, DPs, etc. It is done during production by a script supervisor.

With a pen, they would draw a line perpendicular to the lines of the script from where a shot began vertically down to where a shot would end. As the line passed over a character, it would scribble if the character were not included in the shot, otherwise remain straight if he/she did. A line could be short, covering just a couple lines of the script, or go on for pages. After the line was drawn, the script supervisor would write next to it what type of shot it was, whether close up or wide, etc.

These lines provided a quick visual way to make sure enough coverage was both planned and ultimately filmed. After shooting a scene, that portion of the script would be all marked up. Every line of script would have at least one vertical drawn line covering it. In many cases multiple shots.

The Liner Tool works in a similar way to script supervising lining technique, although, in this case, we're not covering every line of script with a few different shots. We are only interested in marking it with final shots after editing and the completion of production. Each line of script is covered by only one shot in our tool.

When presented with a new script (in the form of a text file), the liner tool starts by reading in the script much like the parser does. But, instead of parsing the script, it copies each line to a data object called LineData. This stores what the actual text of the line is plus data the Liner tool adds to it, such as, is there a cut? What does the shot look like during this line? The liner tool also

invokes the parser on the raw script to get a script object. It uses some of the script object data which will be explained further down.

The LineData is shown visually in a GUI using Java Swing. It has been developed in a way that the user can quickly and simply line the script. The idea is that they watch the movie and use the tool to mark up the script with cuts and shot types, etc. Or, one could even mark up the script with intended shots if no movie existed yet.

For every line of script, the liner tool presents checkboxes and drop down menus to mark information about the script. The most important check box on each line is the cut check box. If it is marked, it is the beginning of a new shot (as well as the end of a previous one.) The shot continues down the script for every line that does not have this check box checked. Every line has also a dropdown box for the type of shot. This is close up, medium shot, wide shot, etc. Usually a shot remains the same throughout its duration, but this is not always the case. Sometimes the camera may move or an actor may move, or both, causing the shot to change. For this reason, the liner tool has a dropdown box for every line regardless of whether there is a cut or not. Similarly, there is a dropdown box for camera motion and one for general composition.

The Liner tool tries to simplify the process. If you select close-up in a dropdown box, it will automatically make the same selection in the dropdown box for all the lines below it up until the next cut. This speeds up the lining process.

The Liner tool also has checkboxes for each line to show whether an object is visible in that shot or not, such as a character or prop. This mimics traditional script lining where you may or may not draw squiggles over a character to show they are in the shot. If the object's check box is selected, again, the same selection is made down every line until the next shot cut is hit for ease of use. The object data is pulled from the Java Script object which was created by the parser. This is the parser's primary contribution to the Liner tool. Only objects that appear in the scene according to the script are given checkboxes in each line. So, if a scene has four characters, those four characters show up on all the lines of that scene in the liner tool. Typically, if a character appears in the beginning of a shot, they are in it until the end, which is why the tool automatically checks all the boxes vertically down until the cut. But, the camera may move or the character may move, making them not visible during the entire shot, so, it is possible to change the boxes to satisfy what actually happens on screen.

The Liner tool stores the data recorded in a Java object with booleans and enums for the above information. The Java Object is called LineData for each line, and there is an array of them covering all lines. This makes writing the data easy, and more importantly, looking up and collecting the data for later use in the Lister Tool.

The data collected by the Liner tool, in the form of an array of LineData, is saved, by converting the object to a JSON to save to an external file. Preferably, the saved file is a zip to save on space, but it can also be the raw JSON for immediate viewing. It doesn't save just the LineData, it also saves the parsed script object as the data goes hand in hand. The Liner tool also provides the capability of reading in a JSON or zip that was exported. This is an important feature in case you want to take a break from lining, save your work and come back to it later.

Primarily, the Liner tool was designed in order to create training sets. However, the training set is the same format as the output of the Lister Tool, that is a script lined by artificial intelligence instead of by a person using the Liner. For this reason, the Liner tool can also be used to show a GUI representation of the Lister tool output.

Building Training Sets

Training sets are a relatively simple concept. We want to get a collection of film scripts and have them lined with data as specified by the Liner Tool. They will be used to train The Lister Tool. It trains by reading in the training sets and creating vectors and probabilities in order to be able to line an unlined script itself.

One script is usually several thousand lines, and even one line provides useful data, so even one script would provide a certain amount of reasonable data to get some result. However, to truly leverage good results, many scripts must be lined. Furthermore, we want scripts from different genres and different time periods to give us better refinement. Films often seem to have different rules depending on whether they're a comedy or drama, and that's something we certainly want to take into account.

There is at least one caveat that should be noted. An actual film may cut or change its shots at any point during a line of the script. And although a cut usually only happens every few lines, it actually is possible that several cuts could happen during a line. In its current form, The Liner tool is making the assumption that only one cut can happen per line, and it doesn't specify exactly where on the line the cut is happening. It is most likely assumed that the cut is happening right at the beginning of the line or perhaps at the end of the line. It many ways, it doesn't really matter, and just knowing the cut happens somewhere on that line is good enough to give us appropriate results. This simplification makes the data easier to create and to process. It is the same with the rest of the line data. It can only change once per line and we don't know exactly where on the line.

The best training set would take a script and the movie and have a user, like myself, manually mark up the script while watching the movie. I'm not a super-human, so, I can't do it in real-time. I need to constantly pause the movie, rewind. Even then, most available scripts are shooting scripts and don't exactly match the finished movies, so I have to make it up a little bit. Getting it close enough will still provide appropriate data.

Training sets could also be created with just the script without the movie, although, in this case, it should be noted that the training set isn't of the actual movie but rather the user's interpretation of the film. This would be the same as if a director or DP was marking up the script before filming it, and even then, it usually doesn't match the finished film. This sort of training set should only be created by people with film-making experience and even then, it should be recognized that it may not provide exact Hollywood data like working with the actual shots of a real and finished movie.

Creating the training sets is a relatively simple process. If you're given a script and movie, it's something that almost anyone can do as long as they can use the simple Liner tool and operate a remote control. Unfortunately, although being a simple process, it is not a quick one. Lining a script can't effectively be done in real time. A two hour movie can easily take many times as long as that. Creating dozens if not hundreds of lined scripts would be beyond me. It is my intention in the next semester to crowd source this project. I will bring in friends and family to line scripts. I will even bring in other students. The Mechanical Turk has even been recommended.

The best way to do the crowdsourcing will be to create a web page with downloads to the scripts I would like lined. Each download box will also have an upload box where the user can upload

the completed lined script. Once one movie has been complete, it will be marked as such so others won't attempt to line it. Of course, I will still need to personally look over every training set and verify that it's been properly completed, either editing it with the Liner tool or edit the JSON file directly in some cases. I have considered creating a raffle to inspire people to participate.

The ultimate goal will be to deliver a training set of a 100 lined scripts, but for the deliverable this semester, I've created one example.

The Lister - Naive Bayes Implementation

The Lister tool is the ultimate goal of this project. The full title would be Shot Lister, as in creating a list of shots. The user inputs in a script in txt format and it outputs a lined version of the script, with cuts, and shot types, etc. For simplicity sake, the initial implementation will output a JSON of the same structure of the Liner Tool. Later versions may have a more pure form with only the shot details themselves but this will be suitable for the initial implementation.

There are two parts to the Lister. The first part is a vector populator. It takes whatever training sets are available and fed into it. The training sets are processed into Vectors. The vectors only need to be created once and can be used for creating the shot lists of multiple scripts. Of course, they can always have more training sets added to them. The vectors contain counts of all the various things that have happened. It has counts of cuts. It has counts of shot type and motion. It also maintains counts of other data such as how many lines there have been since a cut. It can be expanded to included lots of data like how many characters are in a scene or the number lines a character has.

These vectors actually hold all the probabilities needed in order to do a Naive Bayes implementation. They have the probability of a cut itself, which for example would be the the total number of cuts divided by the total number of lines with or without cuts. It would have the number of times three characters are in a scene, divided by the number of lines. But, more importantly, the vectors hold probabilities given other conditions. So, it would have the total count of how many times there were three characters in a scene when a cut happened. Divide that number by the total count of cuts, and you have the probability of there being three characters in a scene when a cut happens. These probabilities might be written like:
P(Cut), P(ThreeChar) and P(ThreeChar|Cut)

The last type of probability shown is conditional probability. Read out, it is the probability of having three character when there's a cut. This means, if you consider all the times there is a cut, what is the fraction of times there are three characters over the total times there is a cut.

Bayes Rule uses probabilities like these to calculate other probabilities. Like the probability of a cut given three characters P(Cut|ThreeChar). Similarly, we might want to calculate the probability of not a cut given three characters P(NoCut|ThreeChar). Naive Bayes would calculate these two probabilities and go with the higher one. So, if the second was higher, it would determine we don't cut. Of course, we would use many more factors other than just whether there are three characters. Ultimately, we want to collect as much data from the parsed script as possible to compare to shot cuts and shot types.

We want to start with a simple set and build from there. Early information will include the lines since last cut (assuming we calculated cut first). We want to include what type of scene object is on each line like action block or dialogue. We'll also probably want to count the number of lines since the scene object switched. Certainly, the number of objects appearing in the scene will matter. Even genre and whether the movie is black and white or color is data that could determine the final characteristics. This list will continue to be expanded.

The second part to the Lister Tool does the actual lining of the script. Its input is the script and the tool utilizes the vectors, at whatever state they are at, in other words, with as many training sets added as possible. It will work with only a few lined scripts as training sets, but the more the better.

The Lister tool starts by reading in the script and asking the user some simple questions such as B&W or color, genre, and year if available. With this input the parser is called and generates a script object. The script object is used to collect and collate various information about each line of the script. It actually works in a similar way to the vector populator in how it gathers counts. Of course, unlike the first part, certain pieces are missing, such as the final output of the cuts and shot types.

The tool moves line by line running the Naive Bayes algorithm with the vectors populated by the training set. For each line, it determines first if there should be a cut or not, then the shot type and

other various information. For this preliminary version, it will populate the LineData object that the Liner tool uses in order to output the final results.

Now, there are actually several Naive Bayes algorithms being run on every line. The most basic is of course the shot cut. There are more complicated ones such as shot type, clean type and motion type. Finally, there is deciding whether each script object that is in the scene should be visible on that line or not. Each one of these can and should use the same data to create probabilities but each one needs to be run separately. However, they do not need to be run completely independently. In fact, the results from one could be used in the calculations for one that takes place afterwards.

Deciding the order that the algorithms should be run in will not be completely trivial and might require some experimentation and thought. Intuitively, deciding the shot cuts first makes sense, but the order after that is up for debate. For example, if a shot is decided to be wide, then it is more likely that more than one character will appear in the shot. On the other hand, if it is decided that more than one character should be in frame, the shot will probably need to be wide to fit them both. So, which should come first? What if a character is visible, and then suddenly they're not? This will most like connect with a camera movement, although it could just be the character walking out of frame. So, these things are definitely connected. Whichever one does come later will certainly need the results of the earlier algorithms included in its probabilities.

Conclusion

This first semester has seen setting things up. Getting things ready to really dig into the project next semester. Many more training sets will have to be collected. The vectors will have to be expanded to include more intrinsic data from the scripts to compare against the data collected in the training sets.

The first implementation uses Naive Bayes. Once we get enough training sets, and have figured out the best way of collating the data into vectors, some other artificial intelligence methods should be considered. Naive Bayes is said to only have about a 70% success rate. Of course, making movies is an art and film-making is often a personal interpretation of a script. It's not an exact science. We can take some comfort in that the results don't have to be perfect because any lined script could be as likely as the next in the real world of film-making. Still, we want to come up with

results that are realistic. We want to avoid cases, for example, where a shot keeps switching back and forth between a close up and a wide shot. We want to avoid having a character appear and reappear in a shot between lines. It's possible such things could happen, but nearly so unlikely that such things should probably be avoided.

But certainly, there are movies that cut very rarely if at all (see Rope, Timecode). There are also movies that cut constantly what seems like every few seconds. So, all these cases should be admissible.

The true thrust of the early part of the second semester needs to be collecting the training sets. A wide variety will be useful. From different genres and different eras. We want movies with lots of cuts and shot types. We want movies with few cuts and a small selection of shot types. From this, we should be able to produce some really nice lined scripts as output.