A Movie Night with Machine Learning

Michail G. Pachilakis
csd3077@csd.uoc.gr

Iordanis P. Xanthopoulos
csd3161@csd.uoc.gr

Abstract

As the number of movies released continuously grows, viewers are flooded with information about several movie productions, making the simplest questions like ”What movie to see tonight?” really hard to answer. Also movie production studios would like to know if a movie could be a commercial success in order to invest money in it.

In this work **i.** we provide a simple question interface so the users can find movies matching on his/her creteria, **ii.** we provide an interface to answer statistic related questions about any movie dataset and **iii.** we provide predictions about a movies imdb score based on Machine Learning.

Keywords Machine Learning; imdb; prediction; spark; statistics; recommendations

1 Introduction

A paragraph of text goes here. Lots of text. Plenty of interesting .test.text.

More fascinating text. Features[1] galore, plethora of promises.

2 Dataset

Some embedded literal typset code might look like the following :

```
int wrap_fact(ClientData clientData,
              Tcl_Interp *interp,
              int argc, char *argv[]) {
    int result;
    int arg0;
    if (argc != 2) {
        interp->result = "wrong # args";
        return TCL_ERROR;
    }
    arg0 = atoi(argv[1]);
    result = fact(arg0);
    sprintf(interp->result,"%d",result);
    return TCL_OK;
}
```

Now we’re going to cite somebody. Watch for the cite tag. Here it comes [**?**, **?**]. The tilde character (˜) in the source means a non-breaking space. This way, your reference will always be attached to the word that preceded it, instead of going to the next line.

3 Implementation

We select to implement three different functionalities. The first one targets mostly the movie production studios and is about movie score predictions. The other two are statistics about the movies contained in the dataset and movie recommendations and they mostly target the simple viewers.

For our implementation we used apache spark and the scala programming language. In the subsection 3.1 is described how we implement the movie predictions and in the subsetion 3.2 are described the implementation of movies statistics and the recommendations.

3.1 Machine Learning

We implementated the movie score prediction using Machine Learning for this reason we select to use apache spark and scala. Spark provided us with MLlib a scalable machine learning library consisting common learning algorithms and utilities. Also it provides us with spark.ml which aims to provide a uniform set of high-level APIs that help users to create and tune a practical machine learning pipelines.

Firstly we created a case class containg 28 fields, as many as the dataset features, and after parsing the initial

dataset we set every feature to it's coresponding class field. We removed from the dataset movies containing commas in their titles because during splitting they generated more features than they should and they generated errors on our comma seperated csv file.

From our initial 28 features we select to keep only those who had meaning for our predictions, so features like "Title", "IMDBLink", "Country" etc. were dropped. We dropped in total 11 features that we thought it would had the least affect in the movie score prediction.

Because our dataset contained a mix of string, double and integer features we had to transform our string features to numeric values. The package spark.ml provides us with the StringIndexer function. StringIndexer converts String values into categorical indices which could me used by machine learning algorithms in ml library.

From our remaing 17 features we had to select only those that would positevly affect prediction of the movie imdb score. To achieve this we correlated our remaining features and excluded those which negatively affected the "imdb_score" feature. Again the MLlib package provided us with the Statistics package which contained the necessary functions for the feature correlation.

In order to create our machine learning pipeline we had to create some transformers to produce our output dataframe. We created a VectorSlicer which takes a vector as input column and creates a vector containing only the features that we set from the original vector. We also created a StandardScaler which is used to scale a vector to a vector which values are in similar scale and a VectorAssembler which is a feature transformer that merges multiple columns into a vector column.

We initialize the Linear Regression Classifier estimator and then chain it with the VectorSlicer, StandardScaler and VectorAssembler transformers in the machine learning pipeline. When the pipeline is used to fit training data, the transformers and the estimator will apply to the dataframe with the sequence defined in the pipeline.

3.2 Statistics and Recommendations

It can get tricky typesetting Tcl and C code in LaTeX because they share a lot of mystical feelings about certain magic characters. You will have to do a lot of escaping to typeset curly braces and percent signs, for example, like this: "The `%module` directive sets the name of the initialization function. This is optional, but is recommended if building a Tcl 7.5 module. Everything inside the `%{, %}` block is copied directly into the output. allowing the inclusion of header files and additional C code."

Sometimes you want to really call attention to a piece of text. You can center it in the column like this:

```
_1008e614_Vector_p
```

and people will really notice it.

The noindent at the start of this paragraph makes it clear that it's a continuation of the preceding text, not a new para in its own right.

Now this is an ingenious way to get a forced space. `Real *` and `double *` are equivalent.

Now here is another way to call attention to a line of code, but instead of centering it, we noindent and bold it.

```
size_t :  fread ptr size nobj stream
```

And here we have made an indented para like a definition tag (dt) in HTML. You don't need a surrounding list macro pair.

> `fread` reads from `stream` into the array `ptr` at most `nobj` objects of size `size`. `fread` returns the number of objects read.

This concludes the definitions tag.

3.3 How to Build Your Paper

You have to run `latex` once to prepare your references for munging. Then run `bibtex` to build your bibliography metadata. Then run `latex` twice to ensure all references have been resolved. If your source file is called `usenixTemplate.tex` and your `bibtex` file is called `usenixTemplate.bib`, here's what you do:

```
latex usenixTemplate
bibtex usenixTemplate
latex usenixTemplate
latex usenixTemplate
```

4 Results

5 Conclusion

A polite author always includes acknowledgments. Thank everyone, especially those who funded the work.

6 Availability

It's great when this section says that MyWonderfulApp is free software, available via anonymous FTP from

```
ftp.site.dom/pub/myname/Wonderful
```

Also, it's even greater when you can write that information is also available on the Wonderful homepage at

```
http://www.site.dom/~myname/SWIG
```

Now we get serious and fill in those references. Remember you will have to run latex twice on the document in order to resolve those cite tags you met earlier. This is where they get resolved. We've preserved some real ones in addition to the template-speak. After the bibliography you are DONE.

References

lalalalala

Notes

[1] Remember to use endnotes, not footnotes!