Introduction to Machine Learning Hackathon – Project description

We chose to accomplish task number 1. In this task, we need to learn to predict the box-office revenue (in dollars) of a movie and its viewer ranking, given some data (a csv file) that describes it.

The data is provided as csv file, with 22 columns and 5659 rows, represents 5659 movies and 22 different features of the movies. The features give a comprehensive view on each movie in various aspects – numerical, like budget, runtime and number of viewers that ranked the movie, and some are semantical – the overview if the movie, the cast and so on. One of the main challenges we came across during the task was to parse the data. In some cases, the data was defective – missing features data for example. Moreover, some features were in JSON format, a format that our crew was unfamiliar with, and the fact that the JSON format was not in a valid syntax made the coping with it even harder.

Our crew deal with those challenging characteristics in the cleaning and preprocessing phase. First, we dropped columns that was not informative to the learning in our opinion – movies id, original title and homepage. Next, we checked in how many movies (=samples) there is a lack in information about a feature of some kind, except from "belongs to collections" feature, that can be empty with no problem. We found out that only 69 of the samples has this lack, so we consider that those samples can be "noise". Because those samples were a small part from the data (little more than 1 percent) of optional noise, we decide to drop them. For the "status" and the language of the movie, after experiments with the data we found out that Boolean factor of "released – True\False" and "the movie is in English – True\False" was the strongest in term of loss. As for the "budget" feature, we found it that its critical for respectable prediction, but about 25% percent of the data has budget set to 0. In order to overcome this obstacle, we calculated the average budget of the rest of the movies and set the movies with 0 budget (that clearly is a mistake) with this amount. This change has significant contribute for our model. In addition, we used the movie release date, and created "dummy variable" for the movie, according to the mouth and decide it was released (for example, a movie that was released in the Christmas of the 2010s has more chance for high revenue then movie from the 1950s that released in regular time of the year).

Our learning model exploits the "overview" feature as follows: we use "TfidVectorizer" - a word frequency scores that highlight words that are more interesting, e.g., frequent in a document but not across documents. In such way, our learning model gives score to every "overview".

The TfidfVectorizer will tokenize documents, learn the vocabulary and inverse document frequency weightings

Taking care of the JSON format columns occurred as follows: first, we handled with the parsing of such kind of features with "eval" and "json.loads" functions. For the "production_companies" feature and for the "production_countries" feature, we made 2 new "dummy variables" – top X, such that a movie got "True\1" if his production company is among the X production companies that made the largest number of movies and equally for the "production_countries" feature. //TODO add crew && cast.

The main considerations that guided our design of the learning system were the Accuracy, training time and number of features. As for the accuracy, we tried to wire the function that predicts a response value for a given observation, such that it will be close as possible to the true response value for that observation. The design of the model included optimization of the running time of the model, along with avoidance from impairing the accuracy – we removed number of features that may not be relevant and significant for the learning - A large number of unnecessary features bog down our learning model, making training time unfeasibly long.

Along the development, we tried various learning models. For start, we tried regular linear regression, and then we checked:

- "Random forest regression" (that uses ensemble learning). loss XXX
- "Extra Trees Regressor": meta estimator that fits several randomized decision trees (a.k.a. extra-trees) on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. Loss:
- "Stacking Regressor" - Stacked generalization consists in stacking the output of individual estimator and use a regressor to compute the final prediction. Stacking allows to use the strength of each individual estimator by using their output as input of a final estimator. Loss: XXX

The learning system we ended up using is Gradient boosting – this model produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

the generalization error we expect in our system is about YYY,