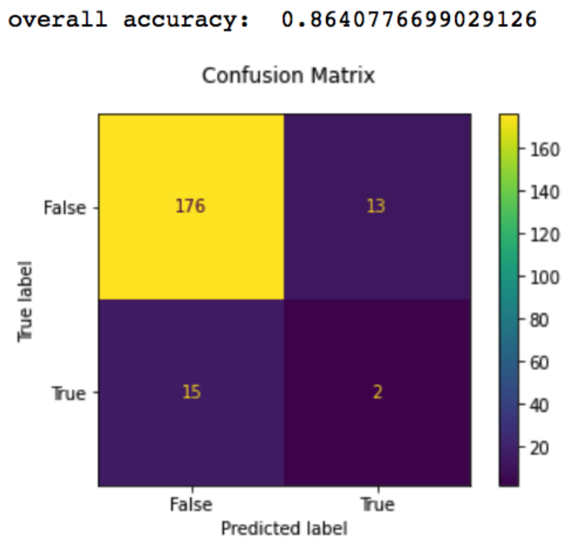
Machine Learning: Final Assignment

Con Óg Ó Laoghaire, olaoghac@tcd.ie

January 25th, 2021

1

I have certainly found it easier to predict the polarity of the review, as opposed to whether or not the review pertains to an early release version of a game. My approach has been to create a bag-of-words, to one-hot encode the most frequent words in this bag, and then to perform deep learning on my one-hot encodings with a Multi-layer Perceptron classifier. One factor that influenced me towards this approach is that there is a large amount of information available. I chose to one-hot encode because I did not see a reason to choose a datastructure that assumes certain words as being more similar to one another - I thought to leave this to the model. A factor in my choice of MLP is that there is a very large amount of parameters, even when considering my decision for only the most abundant tokens to be features - which I did to avoid over-fitting.



```
overall accuracy:  0.8640776699029126
```

Figure 1: Results of MLP approach at predicting early access

My approach works for predicting the polarity of reviews with a greater then 90 % accuracy. What I noticed when I apply this same technique to predicting whether the reviews pertained to early releases is that although I achieved a similar overall rate of accuracy, my module was wrong nearly all of the time for the less frequent scenario, when the review is pertaining to an early release. This can be seen enumerated in Figure 1 above, where early release reviews are miss-flagged 15/17 of the time despite an overall accuracy of 86%.

Outside of my approach being prone to smothering rarer outcomes, I think another reason it works for the purposes of (i) but not (ii) is because the polarity is more so a matter of sentiment analyses, which is quite apparent from the presence of certain key words; e.g. ”great”. Perhaps an approach that would be better suited to (ii) would be one that takes into accounts patterns in the sequences of words: an early release review might stand out based on the presence of sequences such as ”not bad for” and ”will keep an eye”, while the composite words of those sequences alone mightn't stand out as being characteristic of an early version review.

With regards to the accuracy of my approach in part (i), there is scope for improval in cross analysing the filtering of different types of words. To date, I have only done this by eye; that is to say I have removed types of words which I assumed to be inconsequential to predictions, although I am sure a more formal approach would yield a marginally superior accuracy.

2

(i) -

Over-fitting could also be called 'over-learning'. The stage at which the learning becomes over-learning is when the model trains to account for detail and noise in the training data to a degree that negatively impacts the model's performance on new data, i.e. its ability to generalise. A completely over-fit prediction would be one where the prediction is in full accordance with its training data. The graphic representation of this for a linear regression would be a line that passes through all of the training points. Such a graph would have zero variance error, but would probably have an exceptional amount of bias error. Example: Decision trees are inherently prone to over-fitting.

Under-fitting like over-fitting, means the target function is ineffective at generalising to new data. In this case however, the poor-generalisation is the product of inadequate learning. This means it has poor performance on the training data as well as the testing data. It is however, less complicated to solve, so it is not as prevalent an issue. Example: A linear regression that is under-engineered / lacking the appropriate features.

(ii) -

C = [*set of hyper-parameter combinations, or empty if using a non-parametric model*]

```
splits = split_k_ways (data, k)
for ( fold_i in splits ) {
— test = fold_i
— train = splits other then fold_i
— *perform auto feature selection on train*
— for ( c in C ) {
—— for ( fold_j in train ) {
——— validation_set = fold_j
——— model.train(*remaining k-2 folds*)
——— model.predict(fold_j)
—— }
—— get average performance over k-2 folds for parameter combination, c
— }
— train on k-1 folds that yielded the best average performance across inner loop
}
calculate average performance over K folds
```

(iii) -

K-fold cross-validation tunes hyperparameters, but specifically, it tunes them relative to their results across different combinations of data, i.e. relative to the various fold combinations.

(iv) -

KNN is slow in real time as it has to keep track of all of the training data, and find neighbour nodes.

KNN supports non-linear solutions where LR supports only linear solutions.

LR can derive confidence levels about its predictions, whereas KNN can only output labels.

(v) -

KNN is particularly vulnerable to mislabeled data and noise. Either of the aforementioned could completely ruin certain predictions from a KNN model; seeing as the model makes predictions based on a a small amount of the data, this means the effects of noise and mislabelling become apparent in certain predictions.

KNN is poorly suited to larger datasets.