GABRIELIDIS
Hugo

Project Report : Comparaison of different Machine Learning and Deep Learning model To Classify Apps Review

Summary :

I) Description of the problem :

The classification of apps review is an important thematic in the world of mobile apps development. Indeed recent studies have shown that reviews written by the users represent a rich source of information for the app vendors and the developers, as they include information about bugs, ideas for new features, or documentation of released features. In order for apps to keep it's popularity it is essential that developers keep an attentive eyes on those reviews.
However each developers working on a specific field of the apps, there is a need to classify those reviews in different category.

During this project I was interested in comparing the application of various Machine Learning and Deep Learning Model, and see how well they would perform in classifying these apps review.

I decide to choose attribute to each app review one of the four following category : Bugs, Feature Request, User Experience and Ratings.

I choosed this project principally to combine knowledge I got from theoretical study in other course to a practical problem.

II) Selecting the DataSet

In order to train Machine Learning model it is necessary to have an already labeled dataset. For that purpose I used the dataset disponible on the following link :
https://mast.informatik.uni-hamburg.de/app-review-analysis/
This dataset respect the four types of category I defined above, and has been labeled by Master Degree student in the case of a similar study.
The issue with this DataSet is the imbalance it has : the category Ratings and User Experience has too many labeled example comparably to the two other features.
In order to resolve that the state of the art advised us to choose one or several choice between the following : DownSample the set, UpSample the set, use methods such as cross validation, Bagging or Boosting.

In my cases I decided to DownSample the Dataset, which means that I only took 600 labeled example of each class as a Final dataset which means 2400 of labeled reviews in total.

From now on and for each Machine Learning model the training set will consist of : 80% of the initial dataset, the validation set will be 10% of the initial dataset and the testing set will 10% of the initial dataset.

III) Encoding the data

Machine Learning and Deep Learning Model works with array or matrix type of data and since the review are sentences which takes the type of string in python there is a need to transform the data from those string sentence to a vector for each apps reviews. And we want sentence that has the same meaning to be close in the vector space that for ours different models to be able to separate the space in four different regions with one for each class.

One method that has been implemented is TF-IDF Vector, TF-IDF consist of first taking all the words contained in the dataset and make it as a vocabulary.Then we remove the stop words (words like « I », « and » , « or » … that would have impact the classification and would have increased the time complexity of our differents algorithms). After that I lemmatize the vocabulary : lemmatization is the process of grouping together the inflected forms of a word so they can be analysed as a single item,
Then we apply this formula for each words on each review :

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

With that and representing the sentences on the vector space of the vocabulary we will have an array that willl represent each sentences and be able to apply our model on it.
Other methods are known to work better than this one, since we can see several flaw of this methods : it doesn't really take into account some similarity of the words, its true that that good and better will be thanks to classification accounted as the same words but things such as good and magnificent will be considered the same way than good and dogs. So TD-IDF isn't the best way of encoding the review nevertheless I choosed it in order to rate the methods I saw in class.

II) Methods Implemented:

a) Binary classifier :

Binary classifier is the easiest way of classifying and is still used in some request field.
For Binary classifier there is no need of the previous encoding I talk about, binary classifier is about creating for each class a set of keywords that would represent them :

Bugs Keywords : "bug", "fix", « problem »,, »error »,"issue","problem","uninstalling", « issue »,"defect","crash",

Feature Request Keywords : « add », "please", "could", "would", "hope", "improve", "miss", « need », « prefer » « request » « should » « suggest » « want » « wish »

User Experience Keywords : « help","support","assist","when","situation"

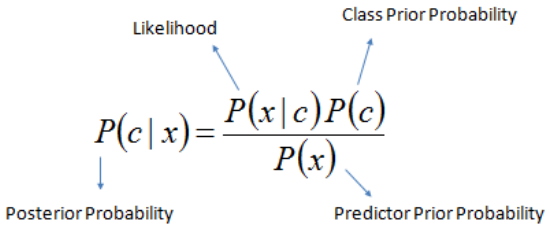
Rating Keywords : « great","good","very","cool","love","hate","bad","worst"

Then for each review we count how much words they have in common with the four class
And give the review the class it has the most words in common (if there is 0 for each we cannot classify there review)

With Binary classier there is no need of validation test and I achieved an accuracy of 0.26 which is really low and just a little bit higher than a random classifier, we can justify it the keywords might have been badly choose and seeing that there a case that append the majority of times : Binary classifier is unable to classify some reviews .

b) MultinomialNB

Naives Bayes will give each reviews a probability of being part of one class using the following formula :



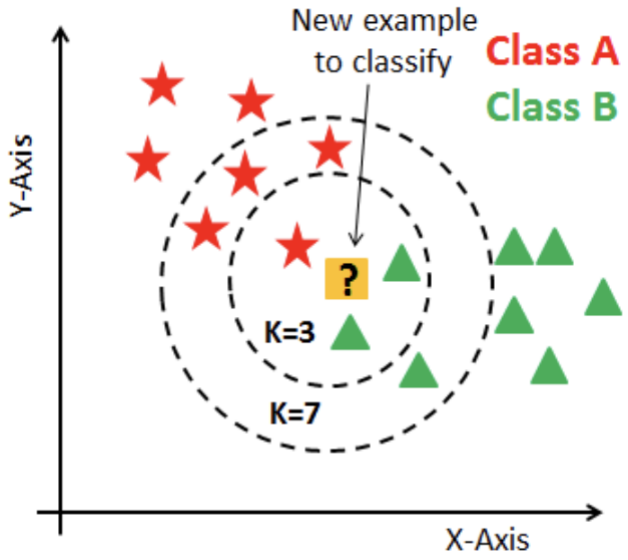
$$P(c \mid \mathrm{X}) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

We compute the probability for each words using the trained set than we for each reviews we compute the probability of being in each class and we attribute the review to the class It had the maximum probability.
Comparably to binary classifier there is no need for a validate set :
Using this model on different training set and testing set I achieved an average accuracy of : 0.65 which is way better than the Binary Classifer but still cannot be considered as really good

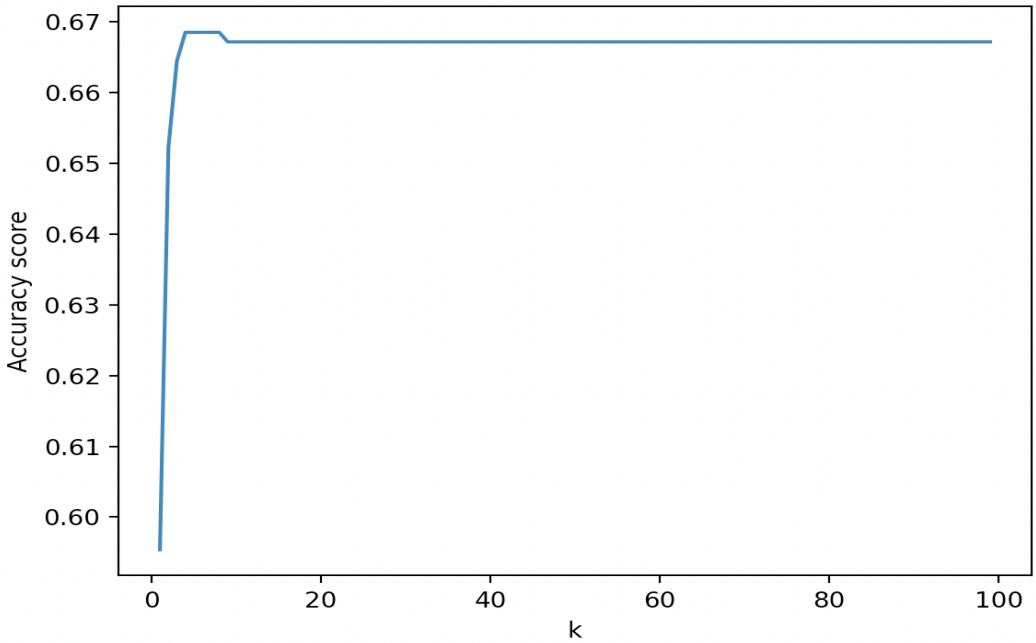
c) K-Nearest Neighbors

K-Nearest Neighbors simplified as KNN works using the following logic, we place the points of the training set In the space and for each points of testing set and validation set we attribute it to the class of majority in majority in its K-Nearest Neighbors using the euclidean distance.

The following picture make it easy to understand the logic of the algorithm :



Since K is a user specified parameters and the performance of the algorithm greatly depends of it's value, we will use the validation set to identify which is the best K using accuracy as a metrics :



We identify K = 13 as the best parameters, we then apply the model with K = 13 on the testing set and achieved an accuracy of 0.66

d) LinearSVC

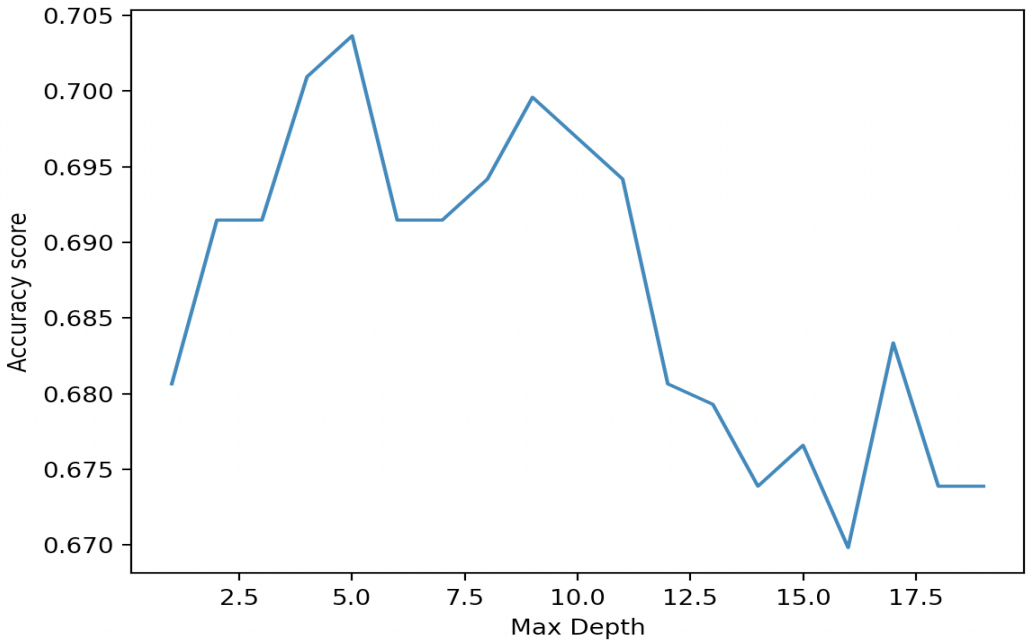
Linear SVM want to find the best hyperplane to separate the differents class, it works on wanting maximize the margin which correspond the distance between the hyperplane and the two class and minimizing the sum of the slack which is the points that have been classified since sometimes the data is not linearly separable. I decided to prioritize the minimization of the sum of the slack and so treat the problem called hard margin SVC.

Here there is not really a need for a validation set, and we achieved an accuracy of 0.665.

e) DecisionTree

A decision tree is a flowchart-like structure in which each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules.

The depth of the decision Tree is a parameters that will greatly influence the performance of the decision Tree so we use the validation set to compare the accuracy that we got on differents decision Tree model using different max Depth parameters :

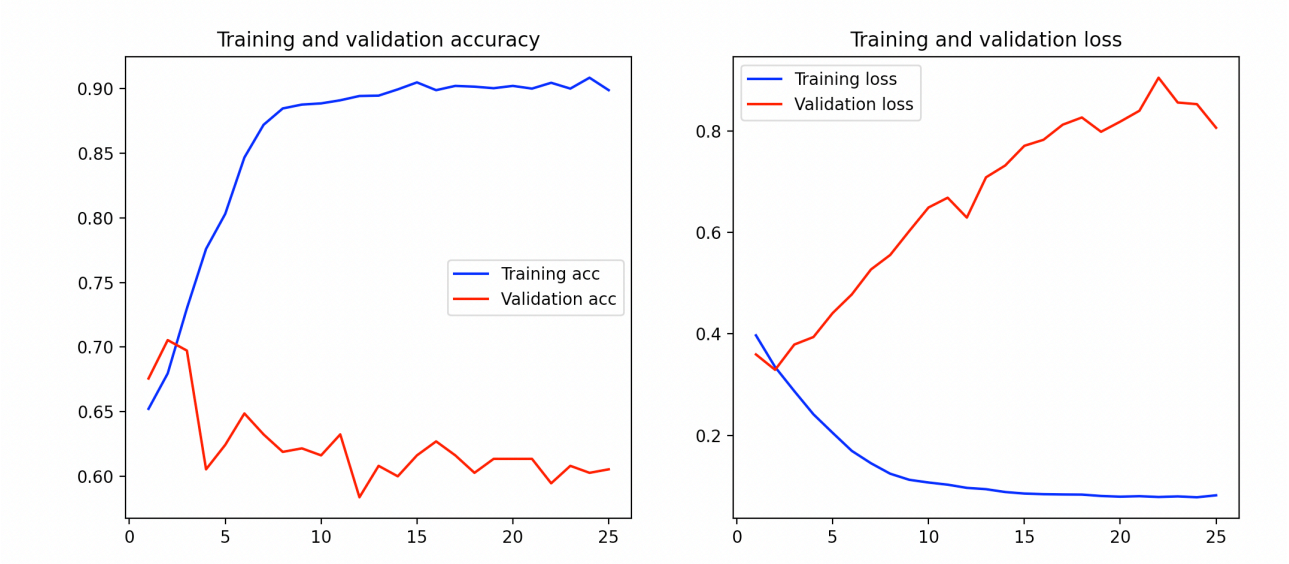


We can identify that Max Depth = 5 is the one that gave us the best parameters, we then apply this model on the testing set and achieve and accuracy of 0.71.

d) Multi Layer Perceptron

I decided to implement the Multi Layer perceptron model using the differents architecture :
An input layer of the size of vocabulary, 3 Dense layer of the respective size of 32,64 and 128 using the activation function « relu », « sigmoid » and « tanh » and finally an output layer of 4 neurons and a sigmoid activation function.

Training the model on 25 epochs and a batch_size of 20 we got the following accuracy and loss graph :



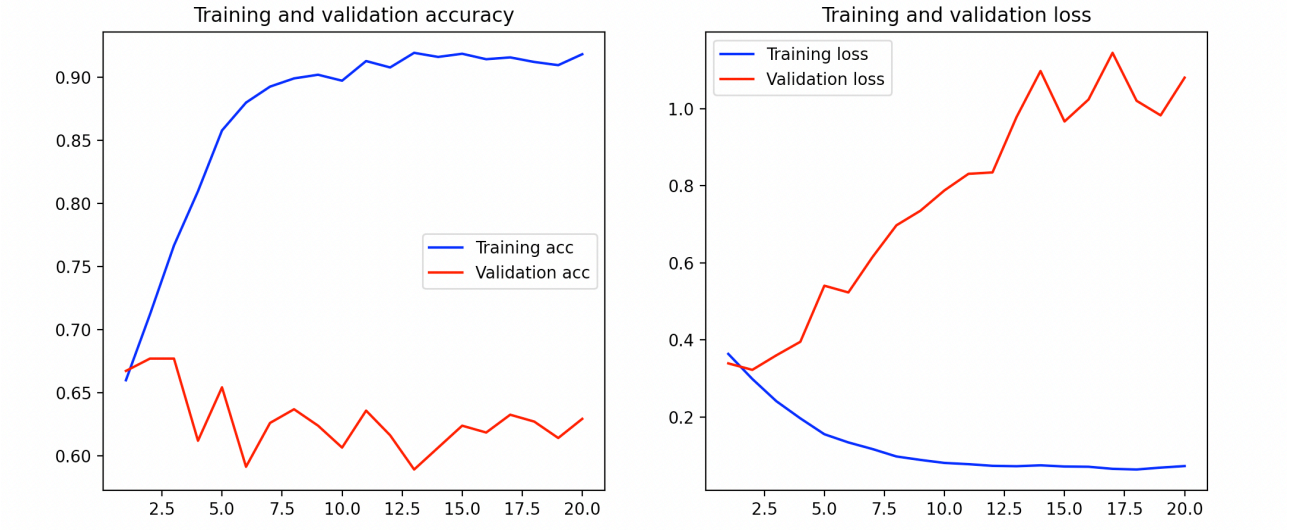
We can see that our model overfit a lot passed and its performance worsen after 2 epochs, we will keep the model trained after 2 epochs as the best one using the callback ModelCheckPointer.

And using this one we achieved an accuracy of 0.70

e) Convolutional Neural Network

For the CNN I used the following architecture an embedding layer, a conv1D Layers with 128 kernel used a Max Pooling layers then 2 dense layers of 64 neurons with a relu activation function and finally an output layers of 4 neurons and a sigmoid activation function.

Training the model on 20 epochs and a batch_size of 10 we got the following accuracy and loss graph :



We keep the best model produced an test it on the training set achieving an accuracy of 0.73

VI) Conclusion

	Binary Classifier	MultinomialNB	KNN	LinearSVC	Decision Tree	MLP	CNN
Accuracy	0.23	0.65	0.66	0.665	0.71	0.70	0.73

For this project I focused on testing various ML and DL model using TF-IDF encoding, it was an arbitrary choice that was motivated by implementing methods I studied in class and confronting my theorical knowledge with a practical experience.

The maximum accuracy I got was :0.73 with the CNN model which isn't really that good. Deep Learning model are known to works well and the reason it wasn't the case here is due to overfit, introducing Dropout layers might have been able to improve results.

VII) Going Further

If I had more time to experiment other things on this project to obtain better result or present the results in a new way I would have :

- Use other metrics that accuracy, in those type of studies usually even if Accuracy is a good metrics other metrics such as Precision, Recall and F1-Score are used. This because for instance Recall might be a good accuracy to detect bugs since we want to be aware of each possible bugs that the user encounter even if their are some misclassification.

- In order to achieve better results I would have liked to experiment :
  - Different type of encoding method, even if Td-if encoding of the data works fairly well other methods such as word2vec encoding or Bert encoding are known to produce better result. Word2Vec works essentially with English language and

- I also would have implemented other Deep Learning Models : if we refer to the state of the art there are more advanced and complex Models that are empirically known to work great with NLP type of problems. In the paper « Attention Is All You Need » published in 2017 by Ashish Vaswani the researcher introduced a model called the Transformer, this fairly complex model is able to resolve the issue of the sequential way of input classical deep learning model encounter when facing NLP problem, and introduced a new type of layer : the Multi-Head Attention which improve the performance of the network.