CS7641 A1: Supervised Learning

Brieuc POPPER
bpopper3@gatech.edu

I. Introduction - Dataset Explanation

I have chosen two very different datasets to work with on this assignement.

A. Simple Dataset : weight&height vs. gender

This first dataset is very "simple": it contains data from Kalahari Kung San people and features 544 entries. More details here : link

The classification task I chose is to **predict gender based on height and weight**. It is clearly feasible with ML as there is a pattern in the data.

This dataset is very interesting for a couple of reasons:

1) **Fast execution times**, more testing and finetuning possible thanks to the small size of the dataset and the fact that the feature space is 2D.
2) **Simplicity** : no problem with the curse of dimensionality, easier to troubleshoot and to explain how the different algorithms perform.
3) **Non-trivial** : simple observations like tall people are usually men exist, but more complex and intricate relationships between weight and height can also be exploited
4) Enables us to **test the code** and get everything working smoothly before moving on to a complex dataset

Pre-processing summary: normalizing height and weight, (no missing data)

B. More complex dataset : Music metadata

This dataset is much more complex, with around 28 000 entries.

For this dataset, the goal is to **predict whether a song has been published before or after the year 1990** based on this features :

1) 16 numbers between 0 and 1 that describe how prevalent the topic is in the song's lyrics (for instance violence, romantic, sadness are topics)
2) 5 numbers between 0 and 1 that describe metrics about the song(its energy or its acousticness for instance)
3) a **hot encoded** genre among : blues, country, hip hop, jazz, pop, reggae, rock

This makes a total of **28 input features**, and the songs are songs from 1950 to 2019 with roughly half of them older than 1990. More details here : link

This dataset is also very interesting to work with because it is **realistic**. This dataset is similar to a lot of real-life datasets for classification, where you have a lot of attributes which you aren't sure how to use or if they're really usable, and you have to deal with the **curse of dimensionality**. However we are going to face **longer execution times**, which means we will have less opportunities to try out very small optimization.

Pre-processing summary: only kept some features from the original database found online, made sure there was no very high correlation between features. No missing data

II. Experimental Methodology

The goal of this assignement is to implement, fiddle around with, and analyze five ML techniques : Decision Trees (DTs), Boosting (done with DTs), SVMs, ANNs and k-NN.

Below is a high level overview of the methodology I followed :

- Find two interesting datasets, understand them
- Setup realistic classification problems
- Pre-process the data
- Shuffle the data into a training set and a test set
- For each of the 5 techniques:
  - Play around with a few different hyperparameter values
  - If something goes very wrong, fix it and keep in mind why it broke. If this is not possible, understand why it's unfixable or very difficult to fix
  - Try to optimize the current algorithm's results, save relevant learning curves and metrics
- Compare and analyze the final results.

III. Individual Results - Simple Dataset

For this dataset, I will track **accuracy**, which is a relevant metric because the dataset has 52% females and 48% males. A trivial model would therefore be able to achieve 52% accuracy.

I will use **cross validation** on every plot (we will be plotting learning curves) that will be part of this document. This ensures we minimize the inherent randomness that is part of these plots that results from choosing an arbitrary split between training data and validation data, by averaging it out.

A. Decision Tree

For decision trees, I am interested in seeing how much **pruning** a single decision tree affects the final accuracy.

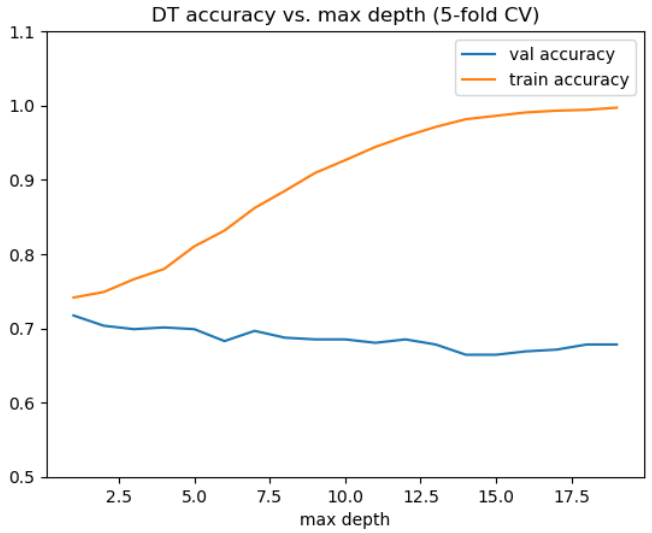

Fig. 1

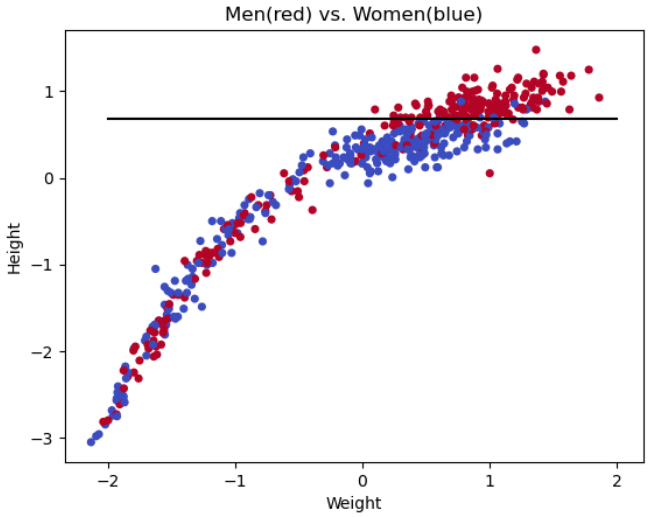

Fig. 2

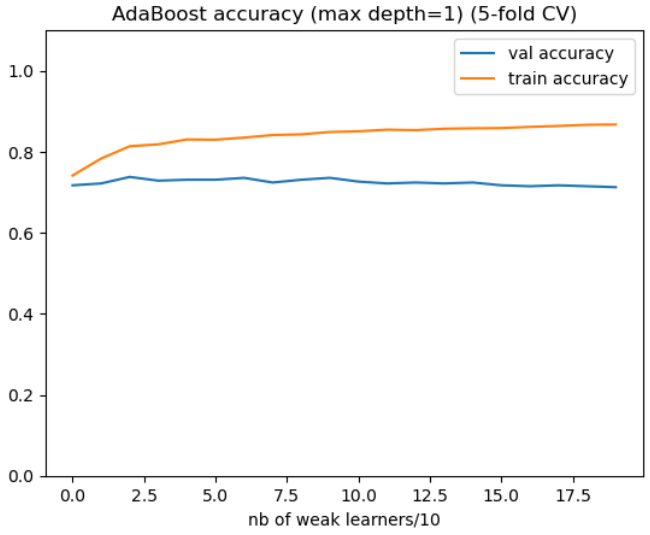

Fig. 3

Recall that pruning is the process of reducing the size or complexity of a decision tree to avoid **overfitting**, which occurs when a tree is too deep and captures noise or random fluctuations in the training data, making it **perform poorly on new, unseen data**. With this definition, I chose to study how limiting a tree's depth (this is the pruning part) would affect it's accuracy. This is displayed in figure 1.

This figure shows a very interesting and surprising fact : t**he tree with the best performance is the tree of max depth equal to 1**. I did a lot of various tests with different hyperparameters and made sure there was no errors in the code, and even so I always ended up by observing the best DT was the one with a max depth equal to 1.

This is surprising as a 1-depth tree is effectively just taking one of the 2 features and comparing it to a threshhold, and doing this remarkably simple operation yields a **78% accuracy** (This 78% was obtained by fitting a DT on the whole training data and testing it on the test data).

To better understand why this is possible, I plotted out the whole dataset, which is showed on figure 2. The **threshold used is displayed as a black line**, and the decision process is : **choose male if you are above the line, and female otherwise**.

Knowing this, this validation accuracy curve is fully explainable, it starts off strong, for a max depth of 2 or 3, it has a similar accuracy than for a max depth of 1. It remains a bit lower which means no very "good splits" were found after that first split (the black line on the figure) or else the accuracy could have gone up a bit.

For too high values of max depths, the accuracy start slowly decreasing, as expected because of the small number of data points and the small number of features : the trees are very **overfit**. This fact is confirmed by the training accuracy curve that goes up to reach 100% for a depth of about 14.

B. Boosting with DTs

Now that we've seen how well a single decision Tree can perform, let's look at what can be done by combining multiple simple decision trees, which will be the weak learners in this boosting algorithm : **Adaboost.**

I've included learning curves for two different cases : one where the weak learner is a DT with **max depth 1**, and one where the weak learner has a **max depth of 2**.

For the max depth 1, looking at what happens for only one weak learner, we get **approximately 75% accuracy** (which is expected given that a single DT of max depth 1 trained on the whole dataset gets 78% accuracy on the test set)

Then we see a slight increase up to a **maximum validation accuracy at approximately 20 learners** and

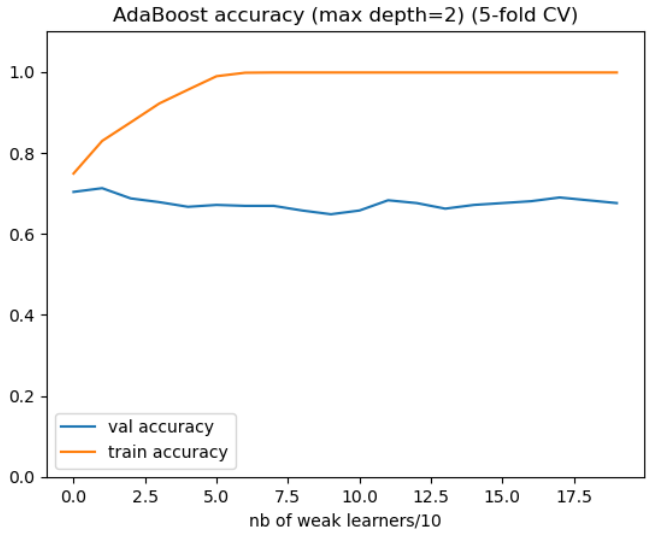

Fig. 4

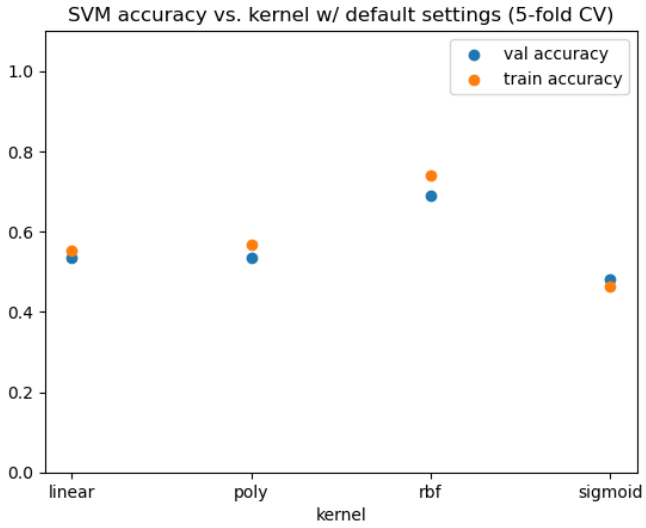

Fig. 5

then we **don't get any significant gain by adding new learners.**

Theoretically, this boosting technique can't overfit if the weak learner is not too complex. However even with this extremely simple weak learner we start to see slight signs of overfitting once we get past 100 weak learners: the training accuracy is going slightly up and as it is the validation accuracy is going down.

This overfitting is a lot more visible (and also more present) on the figure 4 where the training accuracy quickly reaches 100%.

The problem in this specific instance is that **the dataset is extremely small** (only 544 entries) and simple and even a very simple weak learner can cause some overfitting easily.

To illustrate this, imagine a more complex dataset with more entries, more features, and with a hard-to-model distribution of points in each class. If we were to apply a boosting algorithm where each weak learner was a deep neural net with hundreds of neurons, we would expect some overfitting. This is similar to what is happening here.

We manage to reach a **validation accuracy of about 75%** with **20 weak learners of max depth 1**, and this accuracy holds when trying this out on the test data.

We should also keep in mind that because of the very small size of the validation set, a small random fluctuation can appear as a meaningful change of a few percents in accuracy, and so we should be cautious to **not overinterpret a change of just a few percentage points**.

C. Support Vector Machine

To start off I chose to try to see which **kernel** was the best for this classification task. After some comparaison (see figure 5) I chose to **focus only on the RBF kernel**. I tested out a few other hyperparameters aside from the default ones (which are used on figure 5) and I've always found RBF to be clearly ahead on this task.

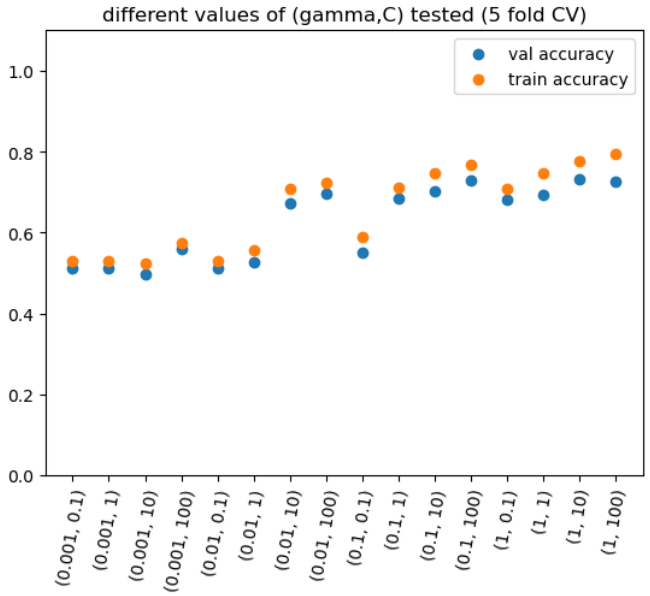

Fig. 6

Two hyperparameters are involved with the RBF kernel : **gamma** and **C**. **A high gamma makes it so only local points matter in the boundary calculation**, so it's more prone to overfitting as it can create complex decision boundaries (as opposed to **a more smoothed out boundary** if you include further away points). If C is large, you make it tend towards having a small margin and so the SVM can overfit more easily.

For the best possible (C,gamma) to have a good validation accuracy we need to try out different values of (C,gamma), and I've done a manual **"Grid search"** (see figure 6) to figure out how much of an increase or a decrease in accuracy would happen with different hyperparameters.

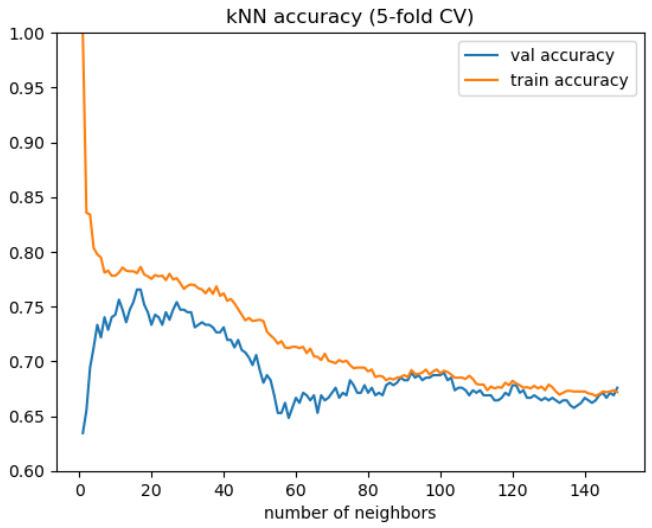

Fig. 7

We can see that these parameters are really important because they can make you go from almost guessing randomly (with say (0.001; 0.1) for (gamma,C) which has a 55% accuracy at best) to having an accuracy on par with the best accuracy obtained so far with other methods (with (0.1; 100) which has a **75% accuracy** approximately)

D. k-Nearest Neighbors

The k-nearest neighbor implemented is very simple : it just takes the k closest points based on a **euclidean** distance and looks at if more of them are males or female. Recall the figure of the 2D distribution of our data (figure 2), we can clearly see that this approach should work with a distance such as a euclidean distance.

This k-NN approach (see figure 7) **works exactly as expected**. First of all for the training accuracy, of course with k=1 it's 100%. Then the training accuracy goes down as we start considering too much neighbors and therefore starting to look at **further away points which are not really relevant**.

For the validation accuracy, we first see that considering too little points is not as good as considering 15 to 20 neighbors. This is interesting, and the fact is that looking at 15 neighbors instead of say 3 is a **tradeoff**.

The 15 neighbors will have some far away neighbors (compared to the 3 closest) which means we can have less relevant points "vote". However it's also possible that having all the 15 point's gender "vote" lets us make a better prediction, because if we're "unlucky" and the 3 points are "misleading" as to indicating what the local trend is, the 15 points will probably indicate it better. This corresponds to the notion of "averaging out noise".

We get our best accuracy with k= 17 neighbors, and we can obtain approximately **77% accuracy.**

50,10	58%
50,50	61%
100,50	59%
10,10	59%
10,50	66%

TABLE I: Hidden layers params vs accuracy (5-fold CV)

E. Artifical Neural Network

After playing around with various architechtures for a deep Neural Network, I obtained decent resulsts with this type of fully connected network.

- **Input** : 2 neurons
- **Hidden Layer 1** : sigmoid activation (5 to 50 neurons approx.)
- **Hidden Layer 2** : relu activation (5 to 50 neurons approx.)
- **Output Layer** : single sigmoid activation neuron

I never managed to get values anywhere above 70% accuracy (on validation/test data of course) with neural networks on this part.

This architecture performs relatively well compared to the other architectures I tried though. Table I shows some sample accuracies for different values of (of neurons in hidden layer 1, of neurons in hidden layer 2)

After some tuning of the **learning rate** most of my architectures showed no sign of improvement past 100 epochs, and so table I was created by testing all the different architectures up to 100 epochs.

Figure 8 shows how for the best architecture which reached almost 70% accuracy the accuracies changed during the training. This training curve is not surprising, the one interesting thing to note is that the model is not complex enough to overfit (as shown by the training accuracy that doesn't continue going up towards 100%) and so it could be the case that we are **underfitting**. However the table showed that some more complex models showed worse performance so we are probably near a sweet spot in terms of parameters for this architecture.

F. Wall times

Finally, we can give the execution time for all of these algorithms. These times are to take with a grain of salt as they depend on the current CPU usage when I start the execution of cells on the Jupyter notebook. Below are the times to create the final classifier with optimal settings.

- Decision tree - *.002s*
- Boosting - *.05s*
- SVM - *.005s*
- kNN - *.001s*

For the NN, it took *11s* to train for 100 epochs the (50,10) network.

IV. Individual Results - Complex Dataset

For this dataset (which is presented in Introduction-B) we will also focus on **accuracy** for the binary classification task.

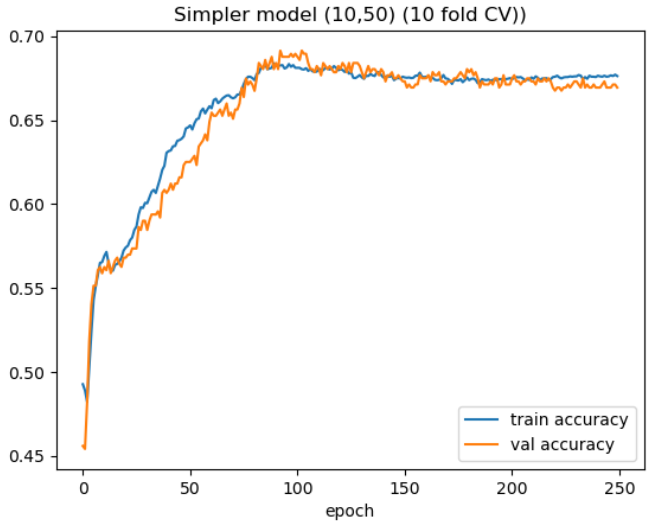

Fig. 8

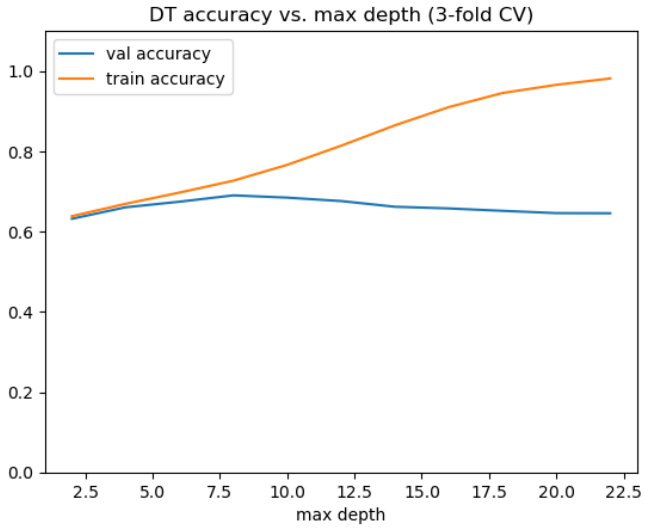

Fig. 9

51% of songs in the dataset are after 1990, and **49% are from 1950 to 1990** which is why we can focus on accuracy only as a metric.

This section will go point by point on the 5 different methods, but will sometimes **reference to arguments already discussed** in the section for the more simple dataset.

A. Decision Tree

As is visible on Figure 9, we see that the there is a very sort of "textbook" relationship between the accuracy and the max depth. We see that there is **overfitting** starting at a max depth of 8, and that the validation curve shows that the **sweet spot is at depth 8 approximately** which is the best validation accuracy, at **about 69%**.

This also shows that very simple trees can be decent at this classification task, which is interesting to keep in mind.

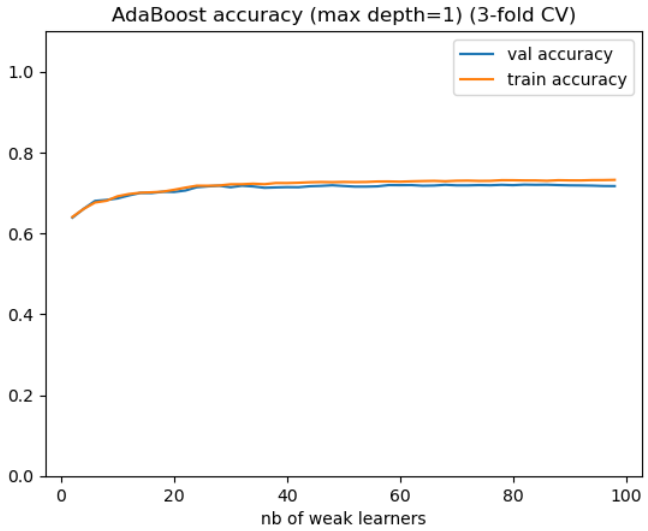

Fig. 10

B. Boosting with DTs

We are now looking at figure 10.

We see that there is **no significant progress after we add more than 40 weak learners**. Up until then it still looks like using more decision stumps (which is just another name for a max depth 1 DT) helps improve our training accuracy.

The plateau at the end means that we reached the limits of what can be done with such a "simple" model on this pretty complex high-dimensional dataset. I checked to be sure what happens at 200 weak learners (in case the curve was just flattening out temporarily), and there's **still no progress compared to 40 weak learners.**

The final accuracy reached is **about 72%**

C. Support Vector Machine

I again started with trying to see what kernel could be good for this task.

With default settings, we get that both poly and RBF kernels get about 72% accuracy, with a linear kernel not far behind.

I chose to **focus on the poly kernel** this time.

As shown by figure 11, we can see that changing the "degree" of the polynomial kernel can cause the SVM method to approch. With this in mind, I tried some different parameters for gamma with **a degree 3 poly kernel** (which seemed like a good choice based on the figure) and managed to get to a 74% final accuracy.

D. k-Nearest Neighbors

This approach (see figure 12) manages to reach **71% accuracy** I tried to see what would happen for more than 40 neighbors, and we can see that the accuracy **starts very slowly going down at about 100 neighbors**, though it remains at approximately 70% accuracy at 400 neighbors.

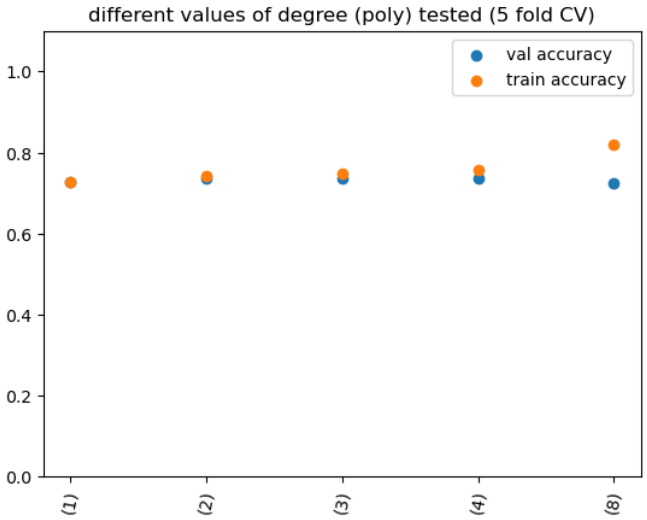

Fig. 11

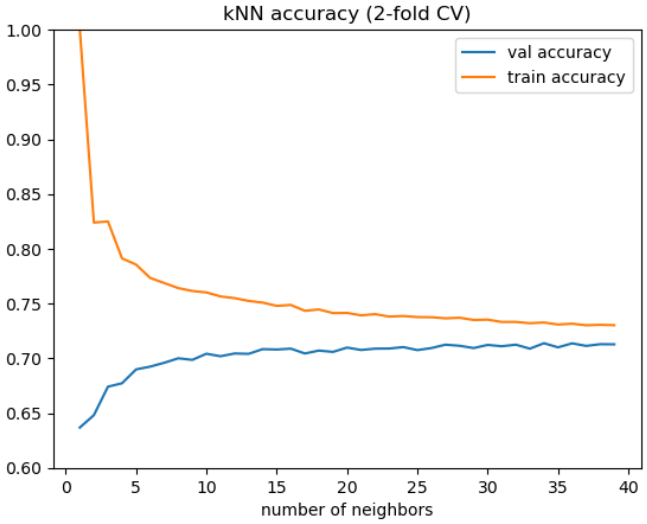

Fig. 12

This is interesting because there is a very large palette of k (the parameter for how many nearest neighbors to consider) that work, but of course **choosing the smallest k possible is better** in terms of speed.

E. Neural Network

After trying a lot of different networks, I could **never get past 73%** accuracy (though to be fair the other methods also had trouble getting past this kind of accuracy)

I used this architecture :

- **Input : 28 neurons**
- **Hidden Layer 1** : relu activation (50 to 300 neurons approx.)
- **Hidden Layer 2** : relu activation (50 to 300 neurons approx.)
- **Output Layer** : single sigmoid activation neuron

The figure 13 shows the **impact of changing the size of the network**. This figure displays the fact that more

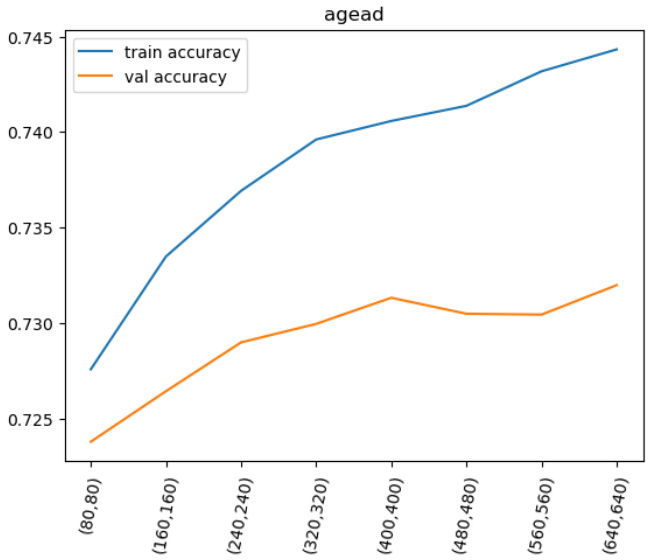

Fig. 13

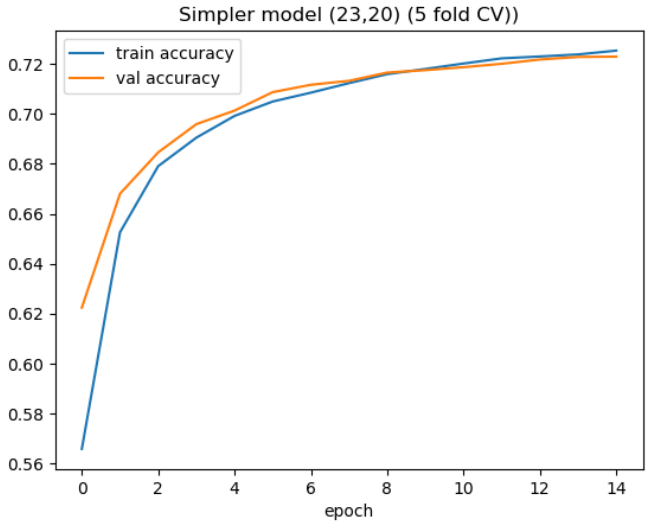

Fig. 14

complex networks - networks with more neurons in their hidden layers - get better accuracies. However this increase in accuracy is extremely small. **The huge increase in computational power and complexity is probably not worth it.** Finally I have included 2 figures that demonstrate how the training happens for 2 networks, with **a more simple network** (figure 14) and a **bit more complex one**(15).

The complex one has **128** neurons on the first hidden layer and **64** on the second, and manages to get up to **73% accuracy** which is about **1%** more than what the simpler model ends up with. The curves are very similar though in terms of convergence speed, which is noteworthy : **one could expect a more complex network to take a lot more epochs to train.**

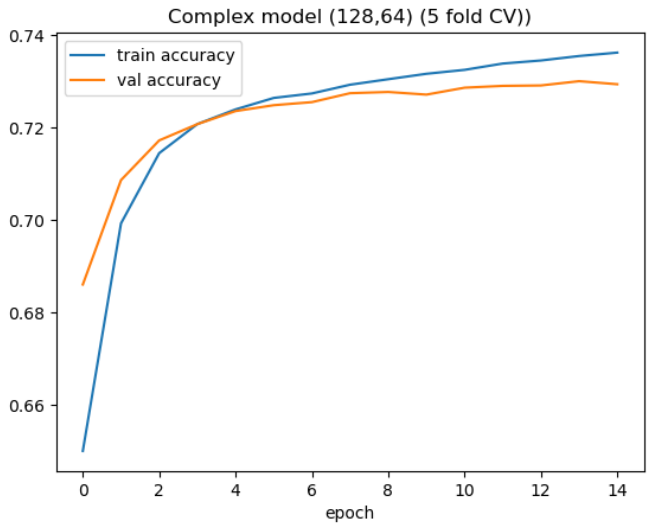

Fig. 15

F. Wall times

These times are for fitting the model on the whole dataset.

- Decision tree - *.696s*
- Boosting - *2.8s*
- SVM - *46s*
- kNN - *3.2s*

For the NN, it took *25s* to train for 100 epochs the (128,64) network.

V. Discussion

It's clear that so far we have discussed some strengths and weaknesses for each algorithm.

A. The best for the simple dataset

A first interesting thing to do is to **sort them by effectiveness for each classification task.** Clearly for the simple dataset, every algorithm was pretty good in terms of accuracy **except the ANN.** Additionnally they all had very small running time, except the neural network. The worst algorithm for this task is probably the ANN which took way too long to compute and didn't match the others accuracy. It is possible that some details went wrong but it's fundamentally not the most relevant tool for such a problem.

Great solutions that are straightforward, fast and provide good scores on this problem include the **decision tree** and the **k-NN method**.

Good but not the best solutions include **boosting method** (but in this specific case a single DT was enough and performed as well), and the **SVM**, whichis a bit more abstract, complex and provides **less explainability** as to how the classification is done.

B. The best for the complex dataset

For the more complex dataset, all methods are viable, except the deep decision tree which doesn't beat the 70% accuracy mark.

However the **boosting method** and the **kNN methods** are **much faster to compute**, which give them a clear edge.

For this task apart from the final accuracy and wall time, the only thing that could interest us is perhaps the **explainablity** of the algorithm, which is how well can we understand how the classification is done.

C. General discussion

Taking a step back, these results are very instructive, but by looking only at these examples we can't fully see the power of every method, and also the full extent of some of these methods' flaws.

It is also important to note that all of the results from the simple dataset are to take with a big grain of salt, because it was **such a simple dataset** that we couldn't beat the accuracy of a simple decision tree with only 1 node ! This means that it's hard to criticize methods such as deep neural networks for their performance on this dataset as they're not made for working on so little and so "simple" data.

Another conclusion is that despite the apparent complexity of the Music dataset, we could c**orrectly classify up to 75% of the songs** using these features, which on their own are only very loosely related to the songs' release date. **This shows the power of ML,** and almost all of these 5 supervised learning methods managed to achieve this.

However, it is perplexing that we could never reach 75% accuracy or more, it's entirely possible that with clever **feature engineering**, more powerful models (different models altogether or different parameters) we could break that 75% barrier.

It is also possible that a part of the 25% misclassified songs are just impossible to classify based on the sole features that are provided to the model (it could be the case that the features are just randomly distributed and can't be traced back to the song's release date).

Finally, a common technique in ML is to **combine various different models to craft a final better model**. This could help **combine the strengths of our different models**, and this could maybe be the key to break that 75% accuracy point.

VI. Conclusion

In this assignement I've had the opportunity to implement and test 5 different supervised learning techniques on **real-world data** !

It's crucial when applying these ML techniques to real-world data to try to tie what we're seeing (on learning curves for example) and to try to tie it back to what we theoretically know, as it's the only way of avoiding common pitfalls.

Finally, I thought I'd list a couple key takeaways of my journey :

- A more complex model isn't necessarily better
- Always start with efficient pre-processing and data visualization
- Try basic models first, you could be surprised by how well they do
- Overfitting happens all the time and can be more or less subtle
- Actual execution times can be very limiting, especially when trying to optimize hyperparameters on large datasets
- Often times you can get a kind of good accuracy not only with one method but with a wide array of very different methods
- Having wildly wrong hyperparameters can make you overlook a potent method and think it's no good
- Learning curves are super important !