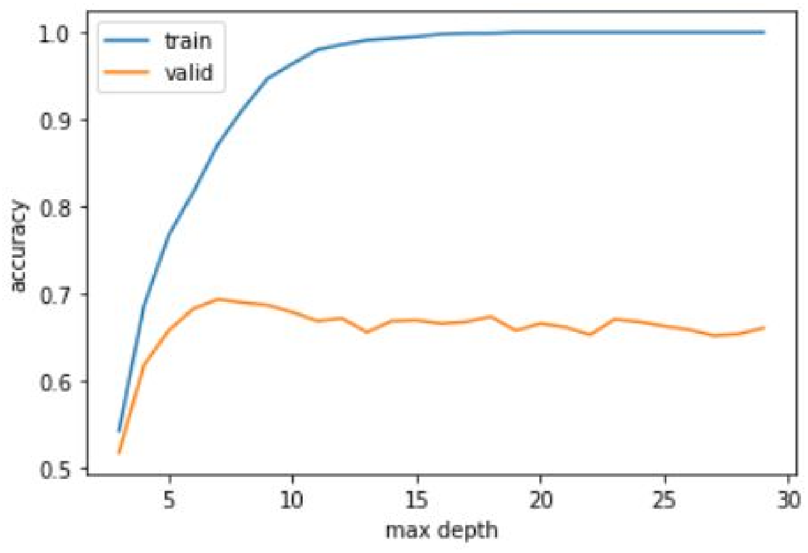
Master 1 AI Paris Saclay
TC0 Machine Learning Mini project
2020 December

Group members:
Mohamed Salem Messoud
Phan Anh VU

Part 1:

Using the dataset Fashion MNIST, we fit a Decision Tree Classifier after transforming the input with a PCA model.
First, we will study the impact of the hyperparameter max_depth.



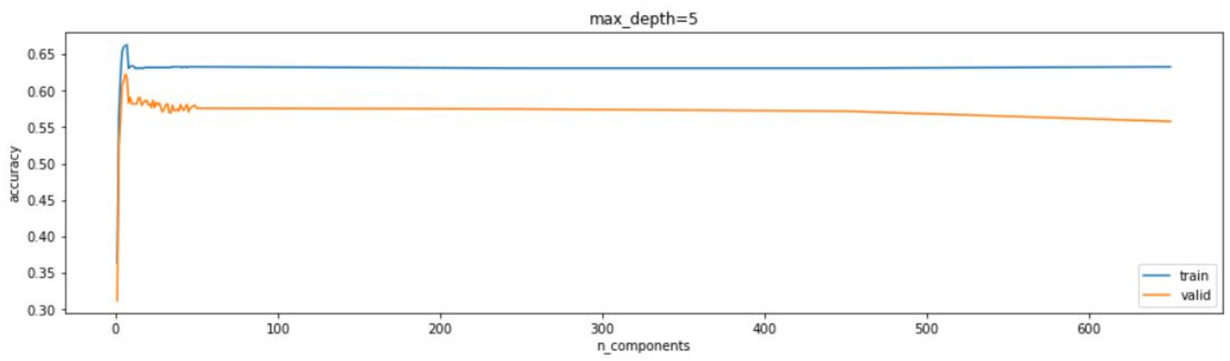
sklearn DecisionTreeClassifier
default hyperparameters
max_depth = 5.

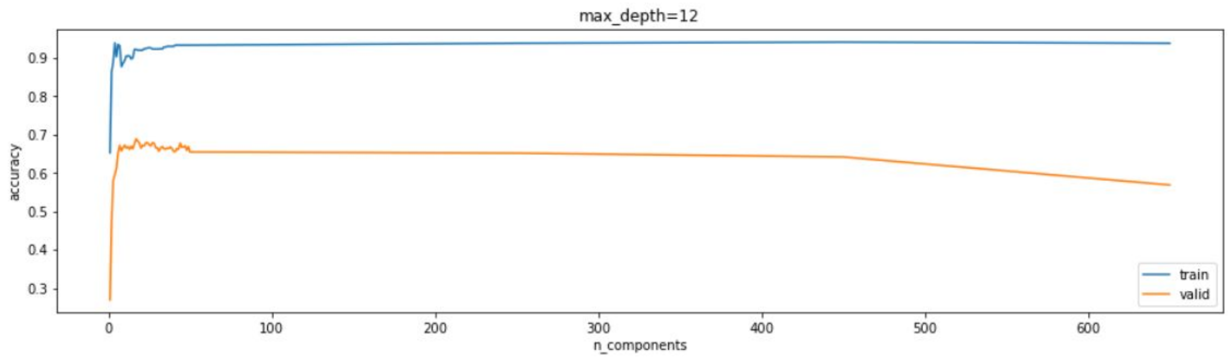
1000 training samples
1000 for validation samples

Best hyperparameter value: 7
Train accuracy: 0.872
Validation accuracy: 0.694

Starting from around max_depth = 5, the gap between train and validation accuracy grows remarkably.
At max_depth = 10, train accuracy is almost 1. Validation accuracy almost remains the same.
This is a sign of severe overfitting.

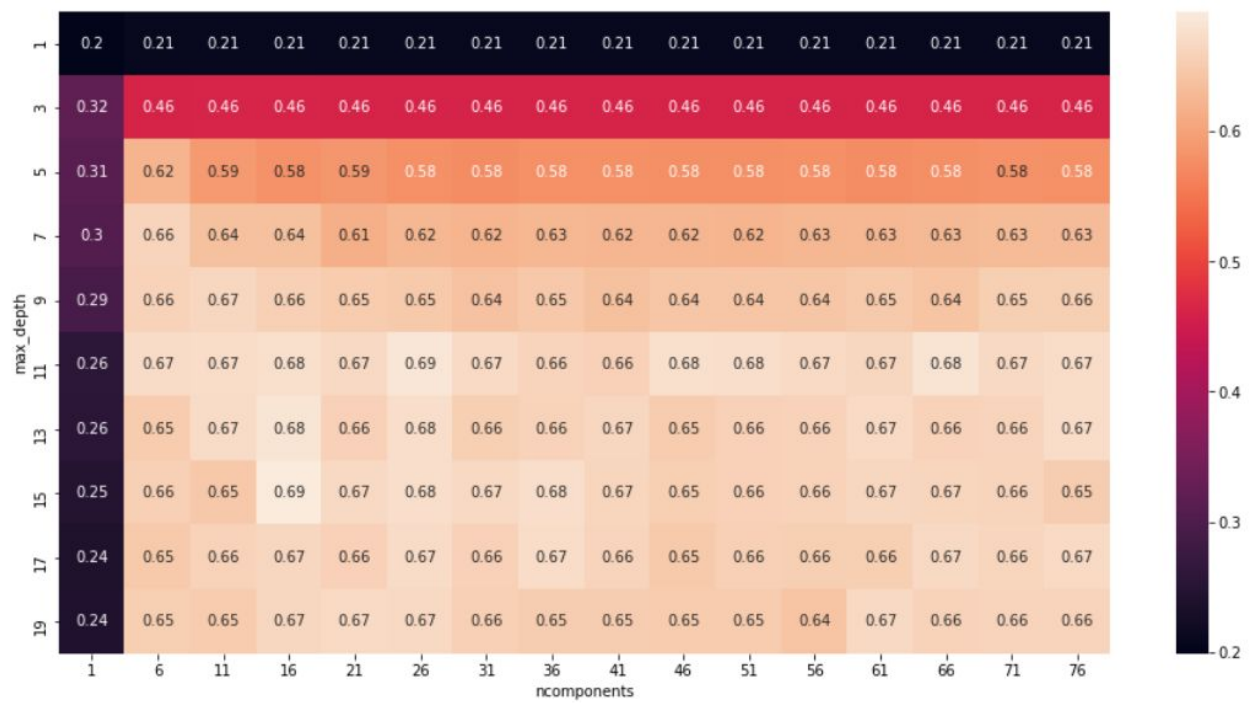
With the same setting, we will now look at the choice of number of principal components.
As max_depth increases, the decision tree model becomes more expressive.
Therefore, the number of components may also rise to match the complexity of the model.
Validation score is better with max_depth = 12.
However, there are signs of severe overfitting. The gap between train and validation score is huge.



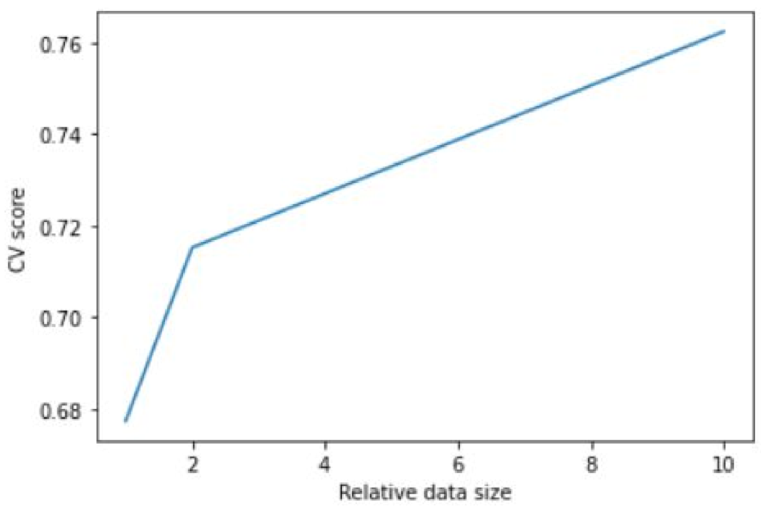


n_components for max_depth = 5	n_components for max_depth = 12
Best hyperparameter value: 6	Best hyperparameter value: 17
Train accuracy: 0.662	Train accuracy: 0.922
Validation accuracy: 0.623	Validation accuracy: 0.689

Now, we will study the impact of the pair max depth and number of principal components together. The heatmap below shows validation accuracy for some combination of the 2 hyperparameters.
We can consider number of components to represent data size, and max depth to represent the model complexity. When number of components, or the amount of data is low, a simpler model having smaller max depth will likely perform better. As number of components or quantity of data grows, the model complexity or max depth tends to rise as well.
The best setting is:
train_scores : 0.978 ; valid_scores : 0.692 ; ncomponents : 16.000; max_depth : 15.000
Train accuracy is 1 or almost, which is an obvious sign of extreme overfitting.
For small number of components, the best max_depth seems to be in the low and middle range.
For large number of components, the best max_depth appears to grow along with the # of dimensions.



With 2000 training samples + 2000 validation samples, and 10 000 training samples + 10 000 validation samples, we observe similar patterns.



Cross validation score for relative scale of data size 1000, 2000 and 10 000.
From what we have seen, getting more data will bring limited benefit.

With 10 000 training samples, the average 5 fold cross validation score is 0.76235, compared to 0.6775 with 1000 training samples. Accuracy improves by almost 0.1, while the amount of data increases by 10.
Scoring on the test set gives 0.7428. The test accuracy is almost the same as validation accuracy. Therefore, this score should be a reasonable estimate for the generalization performance.

Part 2:
In this part, we made our own multi-class perceptron classifier. We used Cross-entropy as a loss function. That might lead to faster training as well as improved generalization. For further information about the derivative of the cross-entropy loss function for the softmax and the update-step refer to the Appendix.

The exponential function grows very rapidly causing a lot of overflow errors even with moderate-sized exponents. To guard against such issues, we transformed the data using the min-max normalization and used the softmax function provided by the Scipy library.

Appendix

The derivative of the **cross-entropy loss function**

$$J = \frac{1}{N}\sum_n^N H(\vec{t}_n, \vec{y}_n) = -\frac{1}{N}\sum_n^N \sum_k^K t_{n,k}\log(y_{n,k})$$

By replacing $y_{n,k}$ with its expression:

$$J = -\frac{1}{N}\sum_n^N \sum_k^K t_{n,k}\log(\frac{\exp(\vec{w}_k \cdot \vec{x}^{(n)})}{\sum_\ell \exp(\vec{w}_\ell \cdot \vec{x}^{(n)})}) = -\frac{1}{N}\sum_n^N \sum_k^K t_{n,k}[\log(\exp(\vec{w}_k \cdot \vec{x}^{(n)})) - \log(\sum_i \exp(\vec{w}_i \cdot \vec{x}^{(n)}))]$$

Either by taking the derivative:

$$\nabla_{w_\ell} J = -\frac{1}{N}\sum_n^N \sum_k^K t_{n,k}\nabla_{w_\ell}[\log(\exp(\vec{w}_k \cdot \vec{x}^{(n)})) - \log(\sum_i \exp(\vec{w}_i \cdot \vec{x}^{(n)}))]$$

We can calculate the two quantities separately $\nabla_{w_\ell}\log(\exp(\vec{w}_k \cdot \vec{x}^{(n)}))$ and $\nabla_{w_\ell}\log(\sum_i \exp(\vec{w}_i \cdot \vec{x}^{(n)}))$

For the first quantity, we distinguish the two following cases ($\ell \neq k$) and the quantity ($\ell = k$).

- if $\ell \neq k$ then $\nabla_{w_\ell}\log(\exp(\vec{w}_k \cdot \vec{x}^{(n)})) = 0$
- otherwise,

$$\nabla_{w_\ell}\log(\exp(\vec{w}_\ell \cdot \vec{x}^{(n)})) = \frac{\nabla_{w_\ell}(\exp(\vec{w}_\ell \cdot \vec{x}^{(n)}))}{\exp(\vec{w}_\ell \cdot \vec{x}^{(n)}))} = \frac{\exp(\vec{w}_\ell \cdot \vec{x}^{(n)}) \cdot \vec{x}^{(n)}}{\exp(\vec{w}_\ell \cdot \vec{x}^{(n)}))} = \vec{x}^{(n)}$$

Or finally

$$\nabla_{w_\ell}\log(\exp(\vec{w}_k \cdot \vec{x}^{(n)})) = \delta_{\ell,k} \cdot \vec{x}^{(n)}$$

For the first quantity, we distinguish the two following cases ($\ell \neq k$) and the quantity ($\ell = k$).

- if $\ell \neq k$ then $\nabla_{w_\ell}\log(\exp(\vec{w}_k \cdot \vec{x}^{(n)})) = 0$
- otherwise,

$$\nabla_{w_\ell}\log(\exp(\vec{w}_\ell \cdot \vec{x}^{(n)})) = \frac{\nabla_{w_\ell}(\exp(\vec{w}_\ell \cdot \vec{x}^{(n)}))}{\exp(\vec{w}_\ell \cdot \vec{x}^{(n)}))} = \frac{\exp(\vec{w}_\ell \cdot \vec{x}^{(n)}) \cdot \vec{x}^{(n)}}{\exp(\vec{w}_\ell \cdot \vec{x}^{(n)}))} = \vec{x}^{(n)}$$

Or in other words

$$\nabla_{w_\ell}\log(\exp(\vec{w}_k \cdot \vec{x}^{(n)})) = \delta_{\ell,k} \cdot \vec{x}^{(n)}$$

For the second quantity, we have:

$$\nabla_{w_\ell}\log(\sum_i \exp(\vec{w}_i \cdot \vec{x}^{(n)})) = \frac{\nabla_{w_\ell}(\sum_i \exp(\vec{w}_i \cdot \vec{x}^{(n)}))}{\sum_i \exp(\vec{w}_i \cdot \vec{x}^{(n)})} = \frac{\nabla_{w_\ell}(\exp(\vec{w}_\ell \cdot \vec{x}^{(n)}))}{\sum_i \exp(\vec{w}_i \cdot \vec{x}^{(n)})} = \frac{\exp(\vec{w}_\ell \cdot \vec{x}^{(n)}) \cdot \vec{x}^{(n)}}{\sum_i \exp(\vec{w}_i \cdot \vec{x}^{(n)})} = y_\ell^{(n)} \cdot \vec{x}^{(n)}$$

$$\nabla_{w_\ell} J = -\frac{1}{N}\sum_n^N \sum_k^K t_{n,k}(\delta_{\ell,k} \cdot \vec{x}^{(n)} - y_\ell^{(n)} \cdot \vec{x}^{(n)}) = -\frac{1}{N}\sum_n^N \sum_k^K \delta_{k,k_{true}^n}(\delta_{\ell,k} - y_\ell^{(n)}) \cdot \vec{x}^{(n)}$$

$$\nabla_{w_\ell} J = -\frac{1}{N}\sum_n^N(\sum_k^K \delta_{k,k_{true}^n}\delta_{\ell,k} - \sum_k^K \delta_{k,k_{true}^n} y_\ell^{(n)}) \cdot \vec{x}^{(n)}$$

As $y_\ell^{(n)}$ does not depend on k, we have

$$\sum_k^K \delta_{k,k_{true}^n} y_\ell^{(n)} = y_\ell^{(n)} \cdot \sum_k^K \delta_{k,k_{true}^n}$$

The terms of the last sum are zero everywhere except when $k = k_{true}^n$, it follows:

$$y_\ell^{(n)} \cdot \sum_k^K \delta_{k,k_{true}^n} = y_\ell^{(n)}$$

$$\nabla_{w_\ell} J = -\frac{1}{N}\sum_n^N(\delta_{\ell,k_{true}^n} - y_\ell^{(n)}) \cdot \vec{x}^{(n)}$$

Hence, the update step:

$$\vec{w}_\ell \mapsto \vec{w}_\ell + \eta\frac{1}{N}\sum_n^N \vec{x}_n(\delta_{\ell,k_{true}^n} - y_{\ell,n})$$