**Documentation for the internship task**

In this document, I will briefly describe my solution to the internship placement task.

**Covertype Data Set**

The data provided half a million observations of tree cover type on 30x30 meter cells and various cartographic/geological/ecological parameters such as: Elevation, slope, soil types, etc.

According to the information in covtype.info, different models such as NN and LDA were already tested on the data, but probably due to technological limitations, the training set contained only 11,340 observations. Nevertheless, they achieved 70% performance with NN.

For this task, I chose a split of 20/80 (training/testing) for the model training. The reason for this choice is that the amount of data is very large and 20% means that the models have to learn 100000 observations, and as the information states, NN could perform decently on 11000 data points. Additionally the 20/80 split makes the models more elastic and secure from overfitting.

**Heuristic**

Upon reviewing the fundamental statistics, including means and the counting of soil types, I determined it prudent to establish a straightforward guideline. Specifically, I opted to compare the elevation of the tested observation with the mean elevation of the nearest type, and classify it accordingly.

**Scikit-learn two simple ML models**

Regarding the two baseline models from the sklearn library, my selection includes the Decision Tree and K-nearest neighbors algorithms. In my estimation, the Decision Tree model is particularly suited to this dataset due to the prevalence of binary features. Furthermore, I believe that the K-nearest neighbors model represents an excellent choice for enhancing the heuristic-based approach.

**Tensorflow NN**

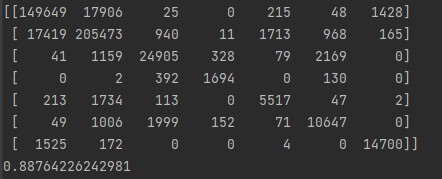
In regards to the neural network model, I have opted to utilize two hidden layers in addition to the output layer. As the target variable in this case comprises integers ranging from 1 to 7, it is necessary to employ the "sparse\_categorical\_crossentropy" loss function. However, this presented a minor issue, as the TensorFlow implementation of the aforementioned function assumes values are sorted and begin at 0, whereas in this instance they range from 1 to 7. As such, for the purposes of model fitting, we utilize Y\_train-1. Conversely, when making predictions, we add +1 to the resulting value obtained from taking the argmax().

Regrettably, due to time constraints, I was unable to undertake an exhaustive optimization of the learning process with the aim of attaining the highest possible accuracy for the neural network model. Nonetheless, I was able to achieve an accuracy of 87%, which I consider to be a satisfactory.

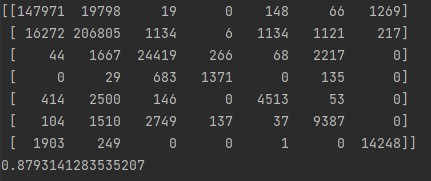
**Accuracy metrics comparison**

All three models demonstrated acceptable performance based on accuracy metrics. It is worth noting that the confusion matrix indicated a high level of accuracy in correctly classifying the majority of cover types, thereby avoiding the accuracy paradox. That being said, the greatest challenge appears to lie in accurately classifying the fourth cover type. Although the models exhibit comparable accuracy rates, the decision tree model appears to be the most effective in this regard, demonstrating the highest level of precision in classifying the fourth cover type.

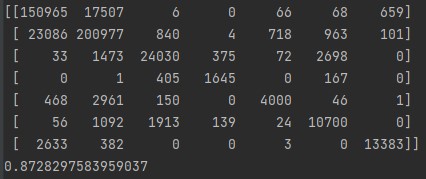
**Decision tree (confusion matrix + accuracy)**



**K-NN (confusion matrix + accuracy)**



**ANN (confusion matrix + accuracy)**



**The code**

The code provided comprises four distinct Python scripts:

1. Loading\_and\_prep\_data.py: This script serves to preprocess the data, including splitting and scaling.
2. Model\_training.py: The primary function of this script is to prepare the model for training.
3. Python\_flask\_REST\_API\_code.py: This script defines a REST API to enable the trained models to be served to users.
4. Call\_the\_API.py: This script contains short code snippets that illustrate example requests that can be made to the REST API. Additional testing samples can be obtained from the covtype.data file.

The code is mostly elastic. The models can be retrained with different parameters (also splitting data into testing/training set can be modified if needed) and launched into REST API.