Recrutation Task

Michał Domagała

Introduction

The Project in question intends to predict the Forest Cover type based on https://archive.ics.uci.edu/ml/datasets/Covertype dataset. It achieves it with 3 models, the last one having 91% Accuracy on Test Data with the potential to go even higher

Structure:

The Solution here provided has been structured in following manner:

All Scripts used to train and house models has been stored in main directory:

1) **Class_tools**.py - provides additional tools, such as saving parameters, step_decay functions, customely made Scaler and so on
2) **ClasTrain**.py consists of functions primarily used to train and grid search parameters of ML and DNN models
3) **MLModelTrainer**.py - main script for training models
4) **ModelComparison**.py - script generating Plots and evaluations of models

Additional Directories are as follows:

1) **Data -** containing covtype.data and covtype.info as well as saved X_test and y_test matrices used for training
2) **Models -** containing saved h5 and .sav models as well as script containing NaiveClassifier Class
3) **Results -** containing plots as well as csv files with evaluation metrics and visualizations
4) **DjangoApp -** Contains necessary files to work as REST API → the view used to fetch models and predictions are contained in views.py

Data

The data supplied regards different parameters of the forest in question and are intended to be used to classify foliage based on that parameters. The variables in question are as following:

1. **Elevation** (meters): Elevation in meters
2. **Aspect** (azimuth): Aspect in degrees azimuth
3. **Slope** (degrees): Slope in degrees
4. **Horizontal_Distance_To_Hydrology (**meters) Horz. Dist. to nearest surface water features

5. **Vertical_Distance_To_Hydrology** (meters) Vert. Dist. to nearest surface water features
6. **Horizontal_Distance_To_Roadways** (meters): Horz Dist to nearest roadway
7. **Hillshade_9am** (0 to 255 index): Hillshade index at 9am, summer solstice
8. **Hillshade_Noon** (0 to 255 index): Hillshade index at noon, summer solstice
9. **Hillshade_3pm** (0 to 255 index): Hillshade index at 3pm, summer solstice
10. **Horizontal_Distance_To_Fire_Points** (meters): Horz Dist to nearest wildfire ignition points
11. **Wilderness_Area** (4 binary columns: 0 (absence) or 1 (presence)): Wilderness area designation
12. **Soil_Type** (40 binary columns: 0 (absence) or 1 (presence)): Soil Type designation
13. **Cover_Type**(**target**) (7 types):   Forest Cover Type designation

The distribution of different classes are not balanced - it has to be accounted for when training models

Cover_Type	count
1	211840
2	283301
3	35754
4	2747
5	9493
6	17367
7	20510

Models

4 models have been trained: Own naive classifier (based on Data Distribution), Logistic Regression, Support Vector Classifier and Deep Feed-Forward Network. They will be shortly described in following paragraphs

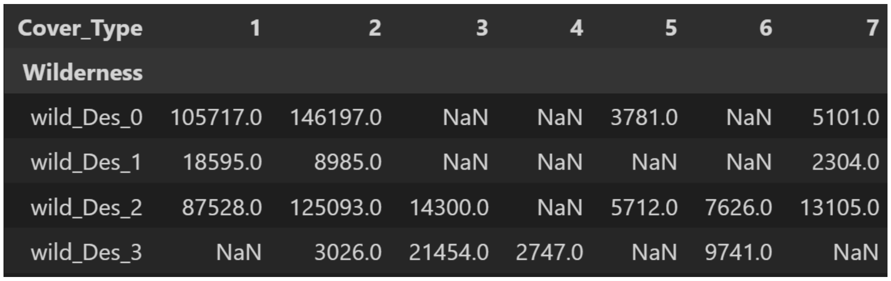
Naive Classifier

Naive Classifier is based on simple observations regarding the dataset. In Supplementary materials to the data one can read that

> *“… for primary major tree species in these areas, Neota would have spruce/fir (type 1), while Rawah and Comanche Peak would probably*

> *have lodgepole pine (type 2) as their primary species, followed by spruce/fir (type 1) and aspen (type 5). Cache la Poudre would tend to have Ponderosa pine (type 3), Douglas-fir (type 6), and cottonwood/willow (type 4).”*

It gives an idea that Wilderness Area (which corresponds to aforementioned places) could be used to effectively differentiate the Cover Types. However, one can see that some multiple types can exist in different areas with similar numerosity. A simple pivot table confirms that:



Cover_Type	1	2	3	4	5	6	7
Wilderness							
wild_Des_0	105717.0	146197.0	NaN	NaN	3781.0	NaN	5101.0
wild_Des_1	18595.0	8985.0	NaN	NaN	NaN	NaN	2304.0
wild_Des_2	87528.0	125093.0	14300.0	NaN	5712.0	7626.0	13105.0
wild_Des_3	NaN	3026.0	21454.0	2747.0	NaN	9741.0	NaN

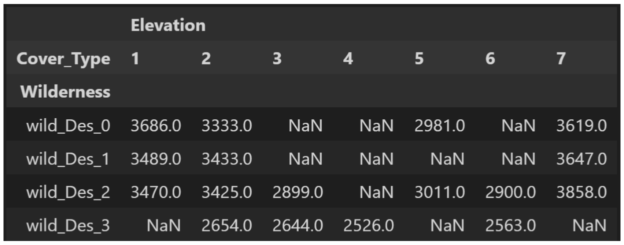
We can see that as described the *Rawah* (wild_Des_0) has the second type most numerous followed closely by the first type. Similarly the *Comanche Peak* (wild_Des_2) and additionally third and seventh being the most numerous. In *Neota* type one is a majority indeed. The *Cache la Poudre* has the third type most numerous and then sixth and second.

To further discriminate, one can look at the Altitude because, as the additional info suggested, it can discriminate between different wilderness areas.

> “*Neota (area 2) probably has the highest mean elevational value of the 4 wilderness areas. Rawah (area 1) and Comanche Peak (area 3) would have a lower mean elevational value, while Cache la Poudre (area 4) would have the lowest mean elevational value. “*

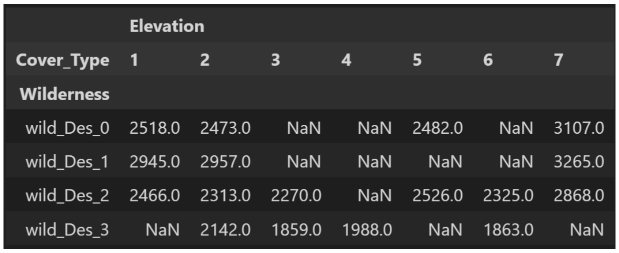
By getting minimal and maximal values of altitude in different Reserves we can see some patterns regarding Foliage type

Table of Max Elevation



	Elevation						
Cover_Type	1	2	3	4	5	6	7
Wilderness							
wild_Des_0	3686.0	3333.0	NaN	NaN	2981.0	NaN	3619.0
wild_Des_1	3489.0	3433.0	NaN	NaN	NaN	NaN	3647.0
wild_Des_2	3470.0	3425.0	2899.0	NaN	3011.0	2900.0	3858.0
wild_Des_3	NaN	2654.0	2644.0	2526.0	NaN	2563.0	NaN

Table of Min Elevation



	Elevation						
Cover_Type	1	2	3	4	5	6	7
Wilderness							
wild_Des_0	2518.0	2473.0	NaN	NaN	2482.0	NaN	3107.0
wild_Des_1	2945.0	2957.0	NaN	NaN	NaN	NaN	3265.0
wild_Des_2	2466.0	2313.0	2270.0	NaN	2526.0	2325.0	2868.0
wild_Des_3	NaN	2142.0	1859.0	1988.0	NaN	1863.0	NaN

One can clearly see that on the first wilderness area, after 3333 meters altitude we tend to have cover 1 and 7 and since type 1 is more prevalent, then it is a way to go in that case. The lower altitude would mean either fifth, first or second type and since we already decided that the first type is above that threshold it is only natural to assume that all lower most likely will be of second type

On second wilderness reserve, altitude cannot discriminate between second and first cover type so it is natural to always choose the first one (as has been suggested by additional info).

On third wilderness area, the altitude above 3470 meters can mean type 7 , then by 3425 can mean usually type 1, lower than that up to 2313 meters will most likely translate to the second type and lastly lower values will correspond to the third type

Lastly on the the fourth wilderness area, by the virtue of numerosity the cover will most likely by Of type 3.

On that heuristics one can build a Naive classifier that will only discriminate based on aforementioned altitude bins and wilderness area type. Note that such a classifier will probably have poor performance. Also the threshold values of altitude are made purely based on data and additional attached info - not on expert knowledge!

Logistic Regression

The first machine learning model from sklearn library that I have chosen is **Logistic Regression** - as it is a prime baseline because of its linear decision boundary that can be easily interpreted and very cheaply and fastly computed. If this classifier proves to be good enough, there is no reason to look for more sophisticated methods.

Training procedure of this classifier involves making a pipeline including customly made StandardScaler - that can Scale only those variables that are of numerical in value (so WildernessType and SoilType one hot encoded variables will be left unchanged) - The second part of the pipeline is aforementioned classifier.

NExt, the gridSearch for regularization parameter C is performed (as it can be substantial in battling unbalanced datasets) using gridSearchCV with Stratified Kfold cross validation (as it is used in multi-class problems) . Then, the best C parameter is used for training the appropriate final model which is then saved using the pickle library in .sav file in the Models subdirectory.

Support Vector Classifier

The second sklearn classifier would be SVC as it has been proven that it can be very powerful but also more explainable than RandomForest-based classifiers. One drawback is that training is very time-consuming. Because of that, although the similar approach as before has been written in appropriate methods, the GridSearchCV takes too much time and therefore, model used in subsequent analysis was trained with randomly chosen parameters C=1 and gamma=0.1.

In similar fashion as in Logistic Regression, the model also has been compiled into a pipeline with customly made Standard Scaler.

DNN Classifier

DNN Classifier has been made customely as a simple 2 layer feed-forward network. The architecture is as follows:

1. **Dense**(1000) (input layer)
2. **BatchNormalization**()
3. **Activation** (Relu)
4. **Dropout** (dropout_parameter)
5. **Dense**(500)
6. **BatchNormalization**()
7. **Activation**(Relu
8. **Dropout**(dropout_parameter)
9. **Dene**(7) with softmax Activation (output layer)

The categorical_crossentropy has been chosen as a loss function as it is most often used in multi-class problems. Because the classes are unbalanced - a weighing parameter was used, computed by using sklearn function *compute_class_weight* - that allows model not ot overfit (which is evident on the Confusion Matrices provided further down)

The Optimizer (Adam or RMSprop) , Optimizer's learning rate and dropout factor have been chosen in Randomized GridSearch as hyperparameters. The metric used to evaluate grid search results was Area under ROC curve (as it can be used effectively in unbalanced datasets and in my experience is generally very effective).

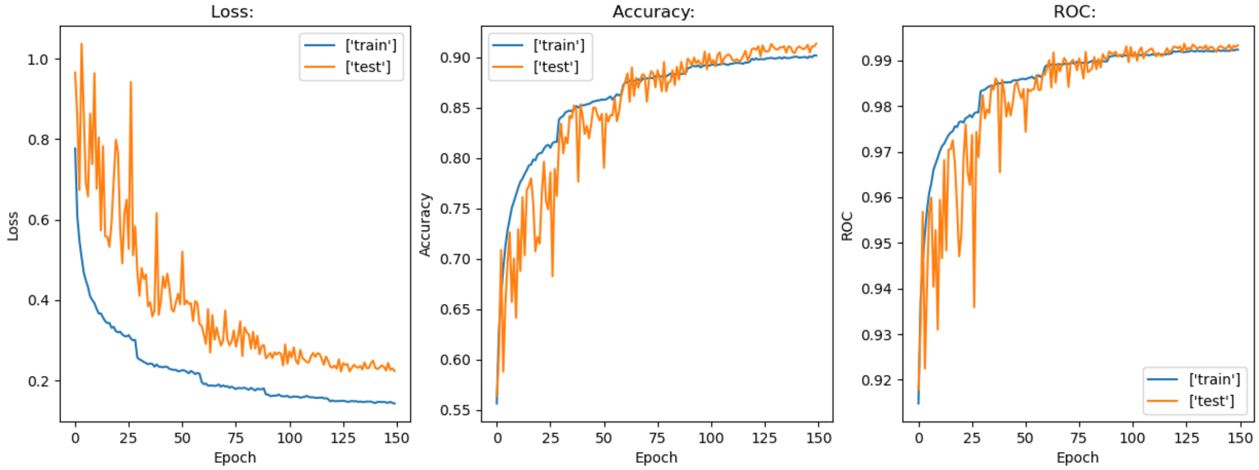
As the data are numerous and are fairly simple, a large batch_size of 256 has been chosen on both grid search and final model training. In Grid Search I have used only 5 epochs (to save time and to determine in first few iterations whether the model is appropriate)

Ater Grid Search has been conducted, model has been trained on 150 epochs and subsequently validated using Accuracy and Area under ROC curve.

Additionally, Early Stopping with patience 25 which monitored maximal validational Area under ROC curve has been used (as to stop training when the model is not sufficiently improving) - this has proven to be irrelevant in current settings - but it shows that a model still has a great potential for learning.

To stabilize the model custom step decay function has been used as Learning rate scheduler (it gradually after every n epochs halves learning rate) -its effectiveness can evidently be seen on training and validation plots

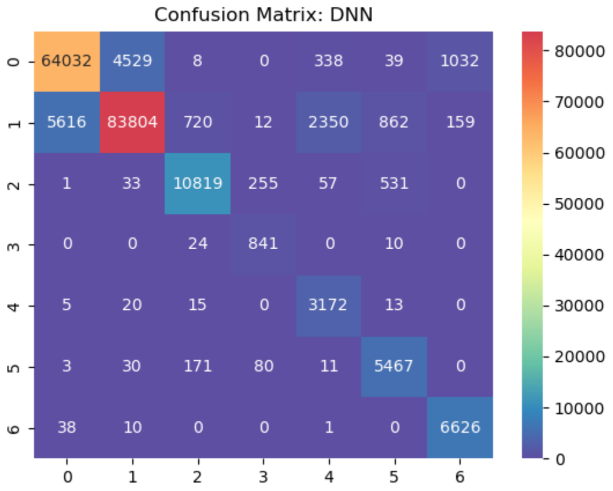
After those steps model has been saved and Loss and Metric plots have been plotted.

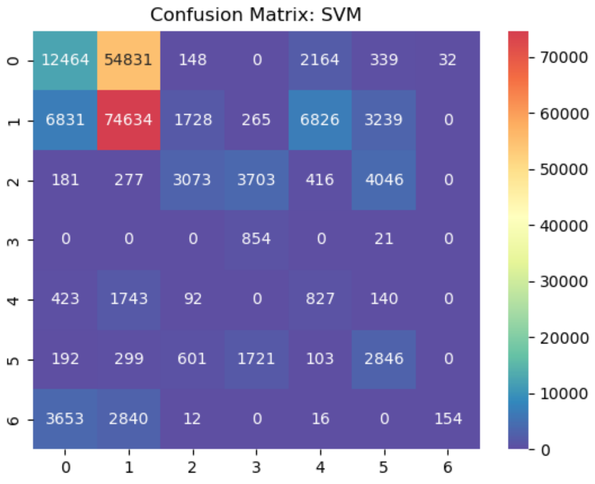


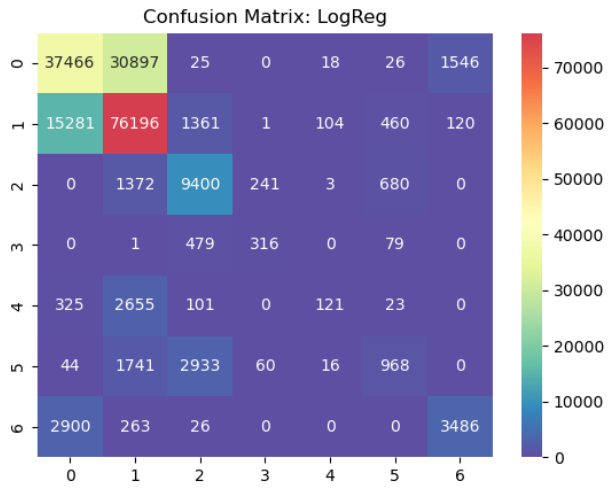
As one can see - Training is very stable and gradual. One can also see despite the flattening (partially by the usage of Learning Rate Scheduler) there is still a room for additional training

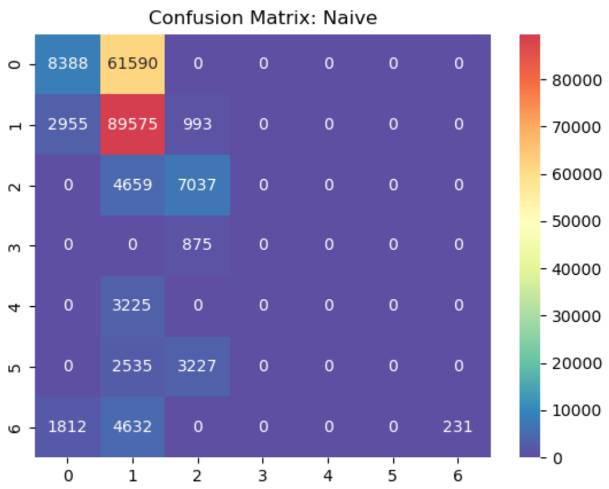
Models Evaluation

To Evaluate models, the Confusion Matrix was used - as it provides great info about distribution of predictions versus actual values.









Visually as one can see The DNN Classifier and Logistic Regression seems to be the most accurate. The SVM has been very time-consuming to train and overall did not perform so great. As is evident by subsequent table:

Method	**precision_score**	**recall_score**	**f1_score**	**accuracy_score**
LogReg	0.6594	0.6673	0.6517	0.6673
SVM	0.5340	0.4947	0.4529	0.4947
DNN	0.9192	0.9115	0.9134	0.9115
Naive	0.5658	0.5488	0.4484	0.5488

Metrics used were precision,recall,f1 and accuracy - as they can accurately show any imbalances in the datasets

One other visualization that would have been in order would be ROC Curve, alas, the time has not been forgiving in that matter. Also the evaluation could be done on Balanced dataset - thus giving definitive proof of the model's imperviousness to balancing issues. The Confusion matrices would also benefit because of this as the differences between good fits and bad fits would be clearly visible

Rest API

The models have been deployed on simple REST API, where by http://*url_name/predict* one can use models to predict by providing all of the 53 parameters that have been used to learn models and name of the models as in [‘LogReg’,’SVM’,’DNN’,’Naive’]