UNIVERSITÀ DEGLI STUDI DI SALERNO

IOT DATA ANALYTICS

MEASURING THE PERFORMANCES OF ONLINE ANALYSIS FOR FOREST COVER TYPE CLASSIFICATION

Professors: Giuseppe Polese, Genoveffa Tortora

Project coordinator: Stefano Cirillo

Student: Michelangelo Esposito – 0522500982

Academic year 2020/2021

INDEX

1. *INTRODUCTION*

Most of the times, machine learning is associated with batch data, meaning that a dataset is obtained from a source of some sort, and then it is used to train a classifier. As a consequence, most machine learning and data mining approaches assume that the elements of the dataset are independent, identically distributed and generated from a stationary distribution. This process is generally adaptable to most "static" circumstances and there are a variety of machine learning models and networks that employ it.

Sometimes, however, the data may not be available all at once from the start, or the hardware on which these machine learning processes run may not satisfy the requirements to run traditional batch learning. Nowadays, we are faced with a tremendous amount of distributed data that can be generated from the ever-increasing number of smart devices. In an IoT scenario, for example, dynamic streams of data are being constantly emitted from countless sensors, and resource-limited devices; in most cases, this data is transient and may be stored for a brief period of time before getting discarded. In this small fraction of time, we are often required to perform analysis operations on the data we receive; therefore, there is the need to adapt the learning process to a workflow that is able to satisfy limited hardware requirements and to dynamically adapt to the continuous flow of data.

Online learning is a method of machine learning in which data becomes available sequentially and is used to update the predictor for future data at each step. It can be data efficient since once an entry has been processed, it is no longer required; therefore, storing previously seen data is not necessary. This approach is also highly adaptable to evolving circumstances since it makes no assumptions about the distribution of the data: as the distribution morphs or drifts, the model can adapt on-the-fly to keep pace with trends in real-time. In order to do something similar with traditional offline learning, one would have to create a sliding window for the data and retrain it every time.

There are various reasons to use online learning over batch learning, among these:

- The whole dataset does not fit in memory.
- The distribution of the data is expected to morph over time.
- The data itself is a function of time (i.e., stock prices).

A very noticeable difference between online and batch learning resides in the training and testing phases of the classifier. Traditionally, there are various ways for evaluating a batch classifier, such as reserving a percentage of the data for testing, or performing k-fold cross validation, a technique by which the data is split into k groups and for each group an evaluation is performed by considering that group as the testing one and the others as the training ones.
In online learning there is no way of reserving a portion of the data for testing purposes, given the volatile nature of the stream, therefore a new approach becomes mandatory. In data stream classification, the most used evaluation technique is the prequential one, where instances are first used to test, and then to train the model.
Finally, something to be really aware of when performing classification over a data stream is concept drift; this the condition due to which the statistical properties of the target variable, which the model is trying to predict, change over time in unforeseen ways. While this represents a problem in batch learning as well, it could pose a significant challenge in online learning since we do not even know the distribution of data to begin with.

In this project we focus on analysing the performances of machine learning classifiers on a dataset containing information about the forest cover type problem. Given a set of cartographic variables and seven different forest cover type classes, our goal is to train different classifiers, in both online and batch fashion, in order to achieve the best accuracy. In the next sections, the composition of the dataset will be discussed in more detail, as well as our approach for the problem.

2. *CONTEXT OF THE PROJECT*

The main focus of this project is to employ online machine learning techniques to analyse the performances of various classifiers on a data stream. This stream does not produce real-time data, but it is generated from a csv/arff file in order to use the online approach.

The platform chosen for the initial analysis is MOA, Massive Online Analysis. MOA is an open-source framework designed specifically for data stream mining with concept drift. It is written in Java and developed at the University of Waikato, New Zealand. It allows to build and run experiments of machine learning or data mining on evolving data streams; it includes a set of learners and stream generators that can be used from the Graphical User Interface (GUI), the command-line, and the Java API.

First of all, both the MOA GUI and the Java API were used to perform three parallel classification tasks on the dataset; these results acted as a baseline for any future run.

At this point, a first comparison among the three classifiers was made, in order to analyse their performances and costs. We then decided to repeat the same process in Python, to further elaborate on MOA's efficiency. Finally, a traditional batch approach was employed to additionally assess the potential gap with online learning.

3. FOREST COVER TYPE DATASET

3.1 OVERVIEW

The dataset used for this study is available for free on the UCI Machine Learning Repository website and its employment revolves around predicting forest cover type from cartographic variables only; no images or other kinds of data are present in this dataset. The actual forest cover type for a given observation (30 x 30 meters cell) was determined from US Forest Service (USFS) Region 2 Resource Information System (RIS) data. Independent variables were derived from data originally obtained from US Geological Survey (USGS) and USFS data. The data was not originally scaled and contains binary columns of data for qualitative independent variables, such as wilderness areas and soil types. A total of over 580,000 entries are present in the dataset, each with the following attributes:

- Elevation in meters.
- Aspect in degrees azimuth.
- Slope in degrees.
- Horizontal distance to nearest surface water.
- Vertical distance to nearest surface water.
- Horizontal distance to nearest roadway.
- Hill shade index at 9am, summer solstice.
- Hill shade index at noon, summer solstice.
- Hill shade index at 3am, simmer solstice.
- Horizontal distance to nearest wildfire ignition points.
- Wilderness area designation.
- Soil type designation.
- Forest cover type designation (dependent attribute). The seven cover types, with the respective number of entries, are:
    - Spruce/Fir: 211840.
    - Lodgepole Pine: 283301.
    - Ponderosa Pine: 35754.
    - Cottonwood/Willow: 2747.
    - Aspen: 9493.
    - Douglas-fir: 17367.
    - Krummholz: 20510.

The primary reason for the collection of cartographic data pertains to terrain mapping. It is ultimately useful for applying topographic correction to satellite images in remote sensing or as background information in scientific studies. Topographic correction is necessary if, for example, we wish to identify materials on the Earth's surface by deriving empirical spectral signatures, or to compare images taken at different times with different Sun and satellite positions and angles. By applying the corrections, it is possible to transform the satellite-derived reflectance into their true reflectivity or radiance in horizontal conditions.[1]

3.2 STATE OF THE ART

The earliest use of the dataset was in a doctoral dissertation. Jock Blackard and Denis Dean at Colorado State University used Linear Discriminant Analysis to obtain a 58% accuracy and an Artificial Neural Network to achieve 70% accuracy. The paper concludes by saying that while the ANN does have its drawbacks, it is still more viable than traditional methods. In 2004, in Systems, Man and Cybernetics an IEEE International Conference publication, Xiaomei Liu, Kevin W. Bowyer of the University of Notre Dame use the forest cover dataset to conjecture a class of problems that are extremely difficult for FFBP NNs, but relatively simple for DTs.

No notable studies involving this dataset in an online fashion were found.

3.3 *DATASET ANALYSIS*

Before employing classification techniques on this dataset, it is necessary to analyse its format, its distribution, its features values, and scan for any outliers. Note that, for the online classification part, we will not consider any balancing or optimization techniques, since we will be assuming that the entries from the dataset are being streamed in real time from a source of some sort.

First of all, the data appears to be well formatted and clean; there are no null values, and all features are numeric. As previously said, there are over 580,000 rows in the dataset and the following graph highlights their distribution in the seven cover type classes:

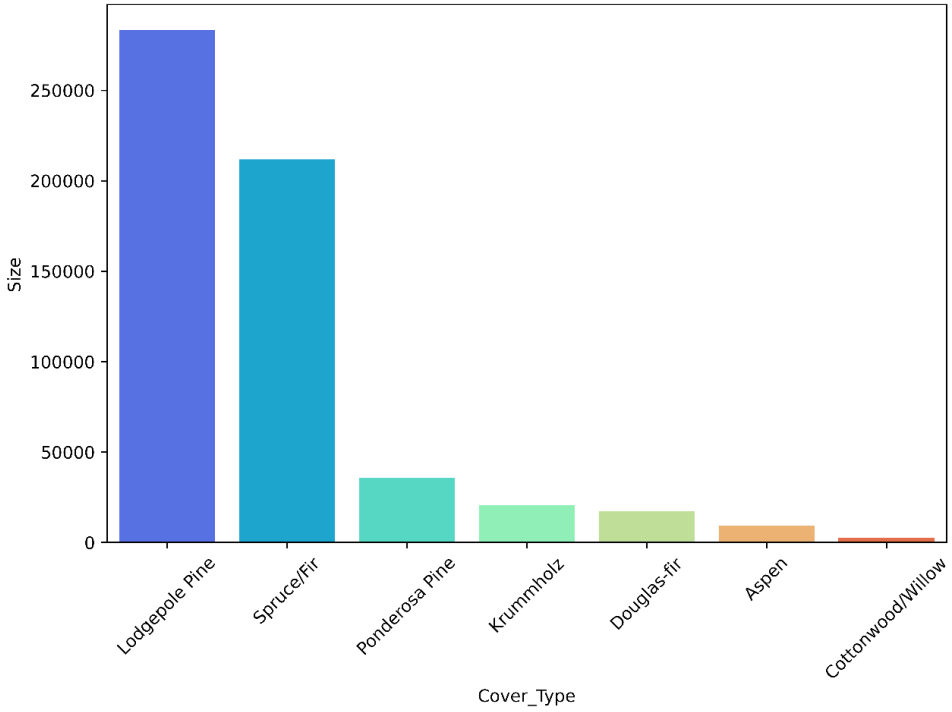


Figure 3.1 Dataset distribution according to the target attribute.

The respective percentages of each class are:

- Lodgepole Pine: 48.76%
- Spruce/Fir: 36.46%
- Ponderosa Pine: 6.15%
- Krummholz: 3.53%
- Douglas-fir: 2.99%
- Aspen: 1.63%
- Cottonwood/Willow: 0.47%

It is easy to notice that the balance across the classes is not quite optimal: while the first two classes, Lodgepole Pine and Spruce/Fir contain a relatively similar number of samples, the other five are underrepresented in the dataset; therefore, some data balancing techniques will be taken into consideration during the development of the project.

We can also observe that the dataset contains some negative values in the Vertical_Distance_To_Hidrology field: this indicates that the nearest source of water is below sea level, therefore these negative values are normal and should not cause concern.

The next thing to look out for is skewness. Skewness is a quantifiable measure of how distorted a data sample is from the normal distribution. In normal distribution, the data is represented graphically in a bell-shaped curve, where the mean (average) and mode (maximum value in the data set) are equal.

The skewness for a normal distribution is zero, and any symmetric data should have a skewness near zero. Negative values for the skewness indicate data that are skewed left and positive values for the skewness indicate data that are skewed right. By skewed left, it means that the left tail is long relative to the right tail. Similarly, skewed right means that the right tail is long relative to the left tail. After computing the skewness for our dataset, some variables appear to be heavily skewed, hence need to be corrected or transformed later. The next skewness representation shows us that, while most of the attribute present minimal skewness, some of the soil types suffer much more from this problem, especially “Soil_Type15”.

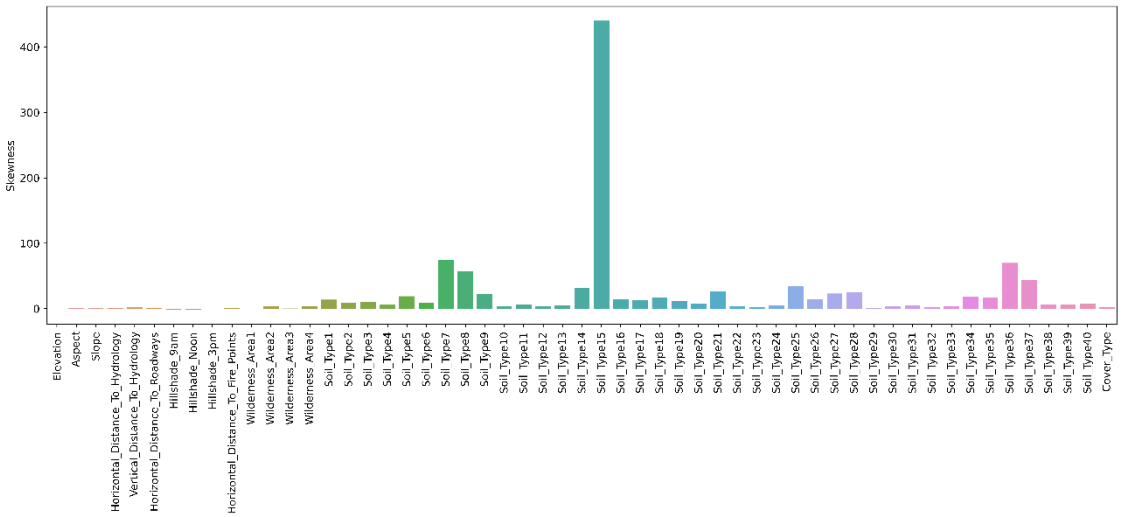
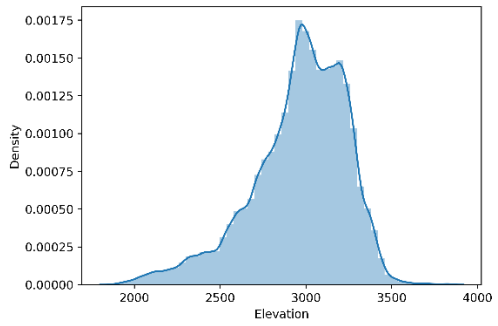


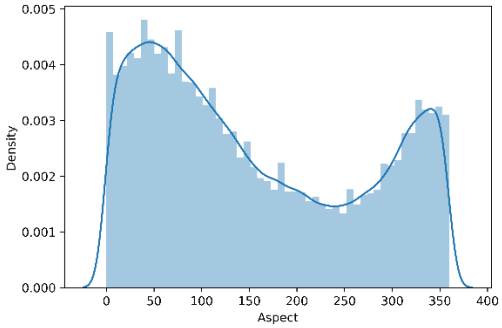
Figure 3.2 Skewness of each feature in the model.

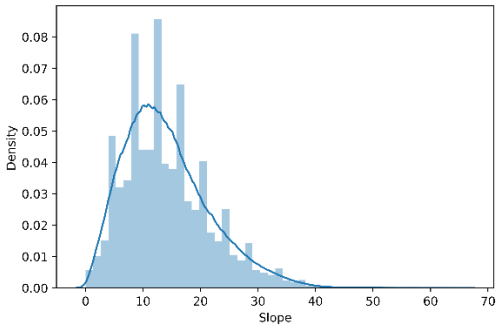
Given the different nature of some of the features in the dataset (i.e., continuous, and binary features), it is worth splitting the dataset into four categories, in order to check the number of value counts within each feature; the considered subsets are:

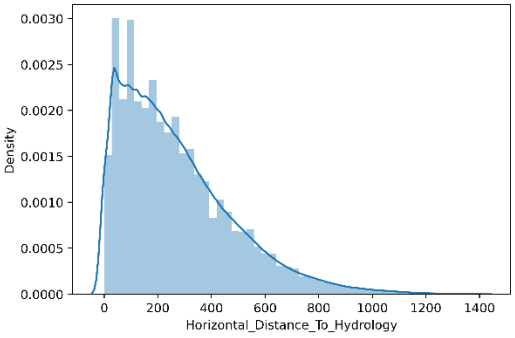
- Continuous data: this includes elevation, aspect, slope, horizontal and vertical distance to hydrology, horizontal distance to roadways, the three hill shades and the horizontal distance to fire points.
- Binary data: the complete set of binary features, including the 4 wilderness areas and the 40 soil types.
- Wilderness areas data: the 4 wilderness areas alone.
- Soil type data: the 40 soil types alone.

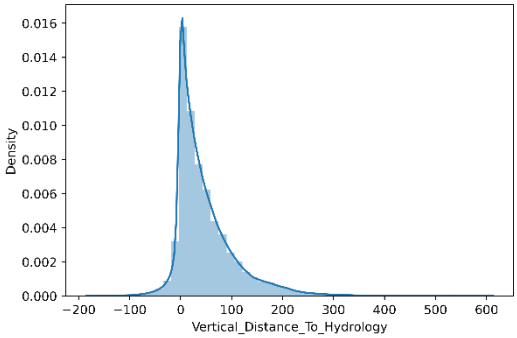
This first set of graphs highlights the distribution of the continuous attributes of the dataset:

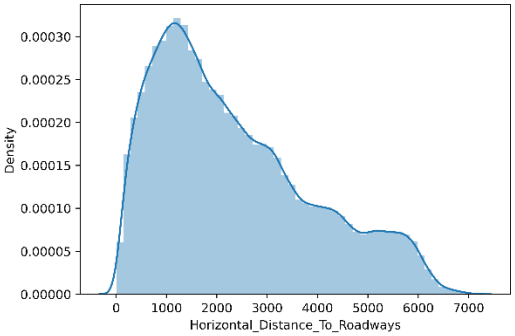


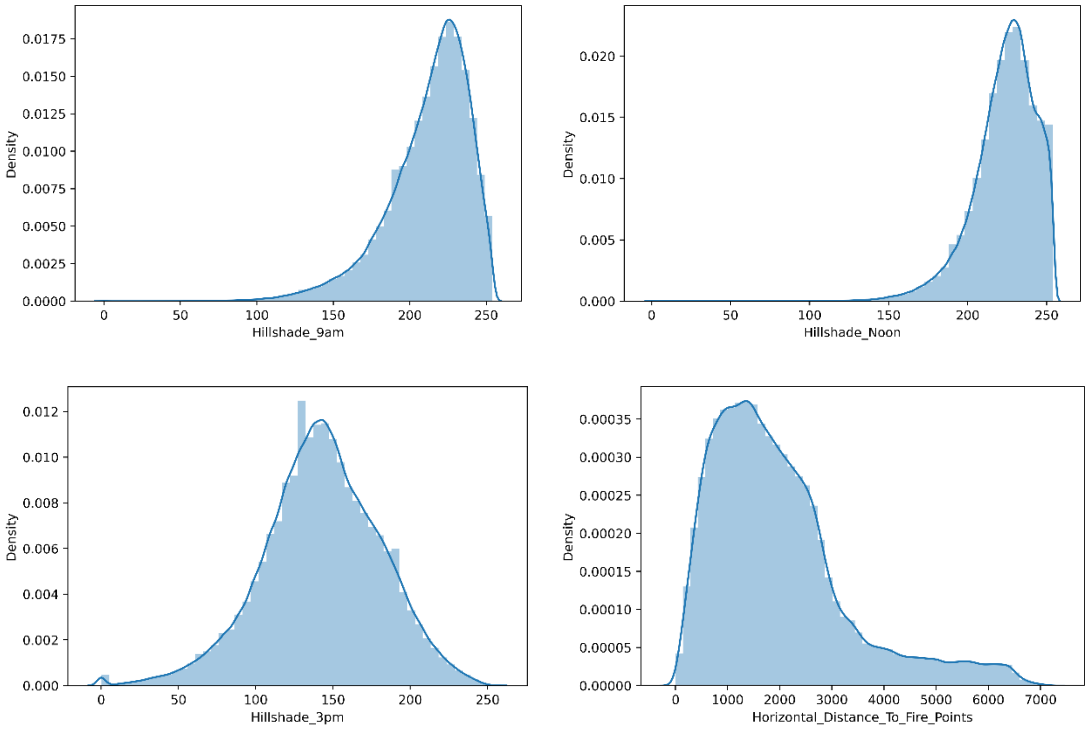












Another interesting feature to analyse is the wilderness area of the trees in the dataset, in relation to each cover type:

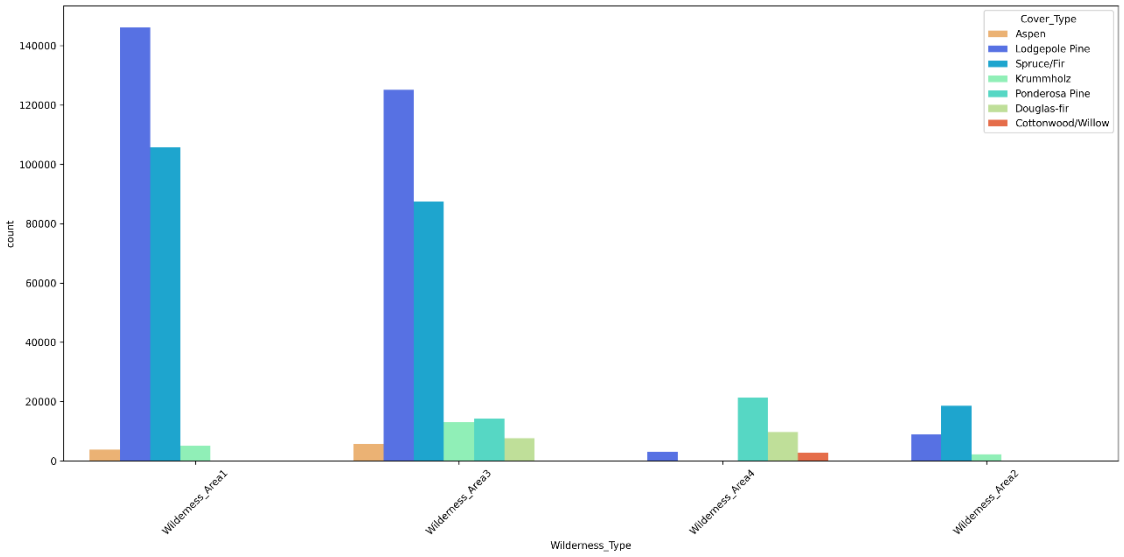


Figure 3.3 Distribution of trees among the four wilderness areas considered.

The distribution of the 40 soil types has also been analysed, finding an unbalanced distribution mostly in favour of the two most present tree types. We do not report the individual graphs for every soil type to avoid cluttering this document; however, the next graph will provide a panoramic view on their distribution:

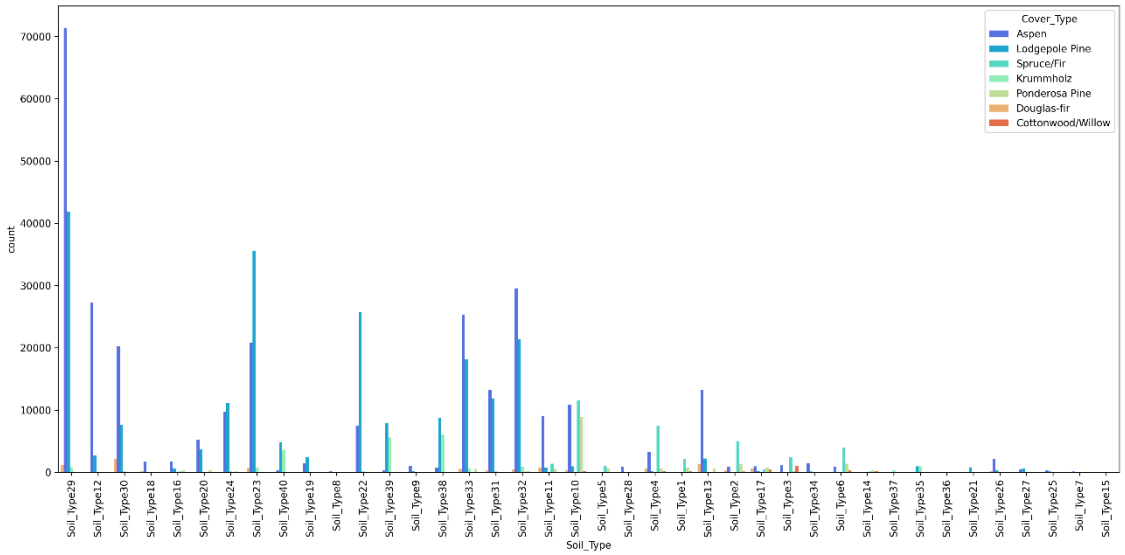


Figure 3.4 Distribution of soil types.

Some soil types are also present in a miniscule number of entries in the dataset; for example, soil_type15 is present in only 3 entries, soil_type36 is present in 119 entries and soil_type8 is present in 179 entries.

A final aspect worth considering is the correlation between the different features. Correlation explains how one or more features are related to each other; it gives us a general idea about the degree of the relationship of two features. Knowing the correlation of the feature of a dataset implies a few things:

- If two variables are closely correlated, then we can deduce one from the other.
- It plays a vital role in locating the important variables on which other variables depend.
- Proper correlation analysis leads to better understanding of data.

High correlation among values, however, can also be harmful for a classifier's performances. We define multicollinearity as a phenomenon in which one predictor variable can be linearly predicted from the others with a certain degree of accuracy. Multicollinearity does not reduce the overall predictive power of a model as a whole; it only affects calculations regarding individual predictors. As a result, we should remove the features which variables presents a higher-than-normal accuracy value.

The next table is the result of computing correlation values among the continuous attributes of our dataset.

Figure 3.5 Correlation matrix of the dataset's attributes.

The most significant correlation scores appear between in the orange cells: Aspect is strongly correlated to Hillshade_3pm, Horizontal_Distance_To_Hydrology is strongly correlated to Vertical_Distance_To_Hidrology and Hillshade_3pm is strongly correlated to Hillshade_Noon. This can give us a general idea of what feature could be removed later on

4. EMPLOYED MODELS

The following models have been employed on the forest cover type dataset:

Online learning:

- Adaptive Random Forest.
- Naïve Bayes
- Hoeffding Tree

Batch learning:

- Random Forest
- Decision Tree
- Naïve Bayes

In this section we will briefly explain how each of them works and what are their strengths and weaknesses.

4.1 DECISION TREE

First of all, a **Decision Tree** is a flowchart-like structure in which each internal node represents an evaluating expression on an attribute (in our case, an example of such test could be: "Is the altitude greater than 1500m?"), each branch represents the outcome of the test, and represents a class label (i.e., "Lodgepole Pine"). When they are being built, decision trees are constructed by recursively evaluating different features and using each node for the feature that best splits the data (in order to minimize the depth, and therefore the complexity of the algorithm traversing the tree).

Among decision support tools, decision trees have several advantages:

- They are very easy to understand and interpret.
- They can be combined with other decision techniques.
- They help determine worst, best and expected values for different scenarios.

Their disadvantages include:

- They are often relatively inaccurate. Many predictors perform better with similar data. This can be remedied by replacing a single decision tree with a random forest, as we will see soon.
- They can be unstable, meaning that a small change in the data can lead to a large change in the structure of the optimal decision tree.
- Calculations can get very complex, particularly if many values are uncertain and/or if many outcomes are linked.

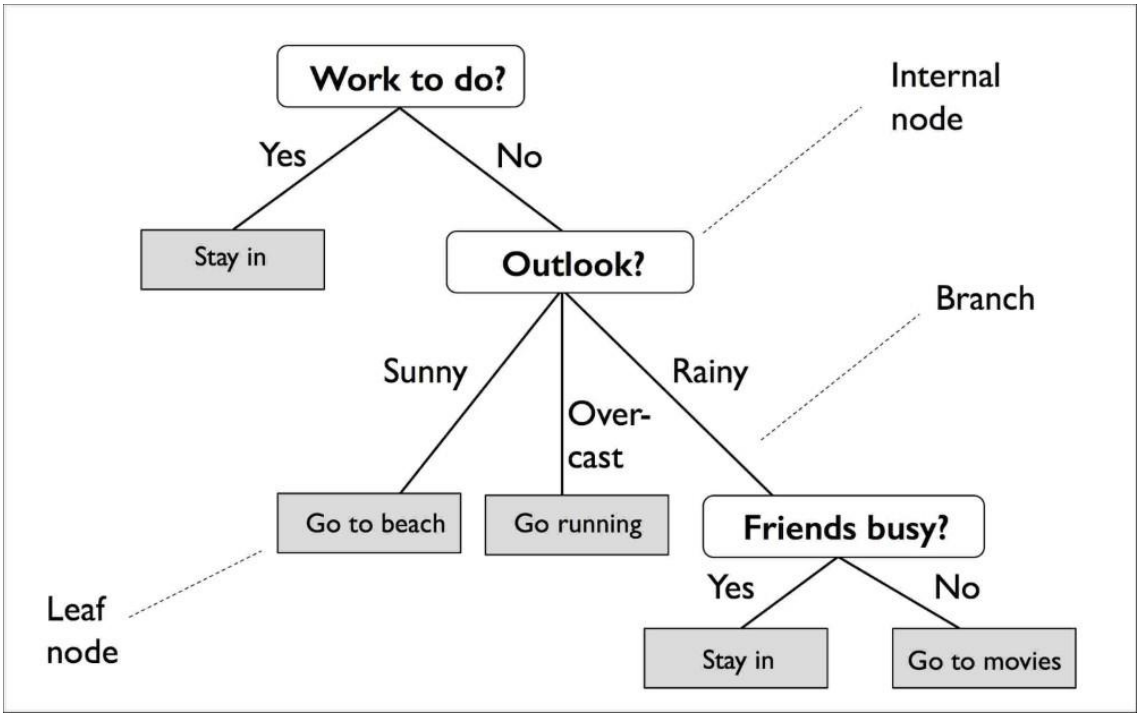


Figure 4.1 Decision Tree example.

4.2 RANDOM FOREST

Similarly, as the name implies, **Random Forests** consist of a large number of individual decision trees that operate as an ensemble: these kinds of models use multiple classifiers to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone. In the case of random forest, each individual tree in the forest guesses a class prediction and the class with the majority of the votes amongst the trees becomes the final prediction. This approach makes it so that the trees protect each other from their individual errors, as long as the majority does not constantly predict the wrong class; while some trees may be wrong, many other will be right, so as a group the trees are able to move towards the correct prediction.

An extension of the algorithm was developed in 2006 and makes use of bagging and random selection of features in order to construct a collection of decision trees with controlled variance. Decision trees are very sensitive to the data that are trained on, as previously stated; random forest takes advantage of this by allowing each individual tree to randomly sample from the dataset with replacement, resulting in different trees. This process is known as bagging, or bootstrap aggregation.

While generally performing better than singular decision trees, random forests lack their intrinsic interpretability which normally allows developers to confirm that the model

has learned realistic information from the data and allows end-users to have trust and confidence in the decisions made by the model. Furthermore, employing a random forest classifier with a non-trivial depth can significantly slow down the classification process.

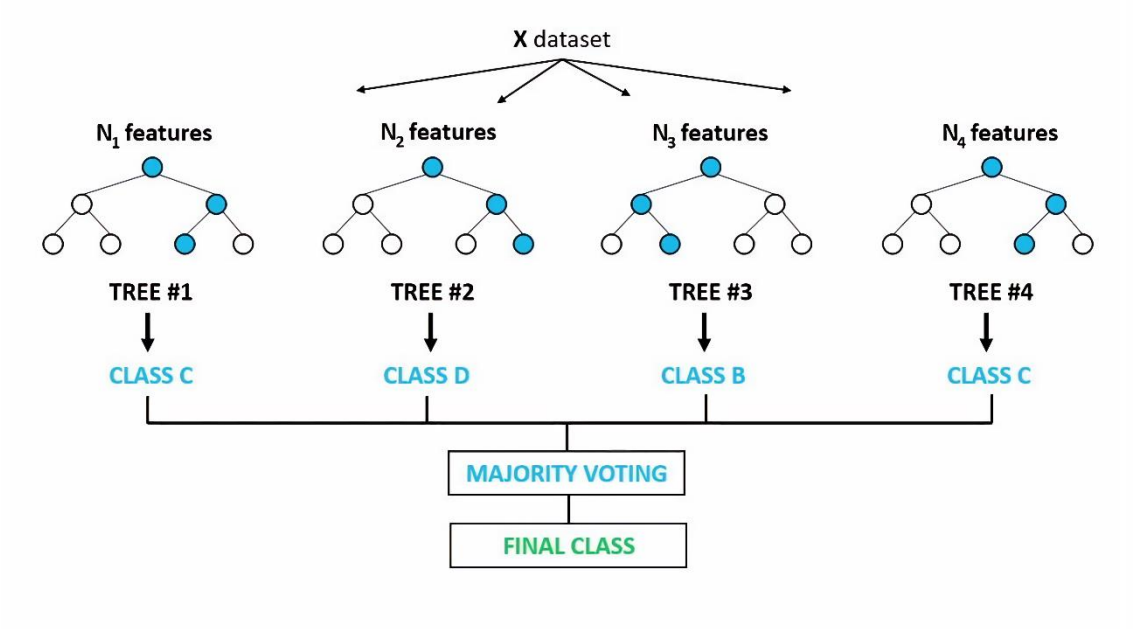


Figure 4.2 Basic decision process behind the Random Forest algorithm.

4.3 HOEFFDING TREE

Considering the data stream context, where we cannot store all the data, the main problem of building a decision tree is the need to reuse instances to compute the best splitting attributes. The **Hoeffding Tree** is an incremental, anytime decision tree induction algorithm that is capable of learning from massive data streams assuming that the distribution generating examples does not change over time. It produces decision trees which are similar to traditional batch learning method; Hoeffding trees and decision trees are asymptotically related. The HT algorithm is based on a simple idea that a small sample can be often sufficient to choose an optimal splitting attribute. The key point is that traditional batch learning methods also generate decision trees based on splitting attributes. A node is expanded as soon as there is sufficient statistical evidence that an optimal splitting feature exists, a decision based on the distribution-independent Hoeffding bound.

4.4 ADAPTIVE RANDOM FOREST

Remaining in the context of evolving data streams there is no random forest algorithm that can be considered state-of-the-art in comparison to bagging and boosting based algorithms [2]. In contrast to the traditional approach, **Adaptive Random Forest** includes an effective resampling method and adaptive operators that can cope with different types of concept drift without complex optimizations for different datasets (hence Adaptive). The adaptation process depends on an appropriate bootstrap aggregation process and limiting each leaf split decision to a subset of features. The second requirement is achieved by modifying the base tree induction algorithm, effectively by restricting the set of features considered for further splits to a random subset of size *m*, where $m < M$ and *M* corresponds to the total number of features. To address the first requirement, given the Online Bagging algorithm, proposed by the authors in [3], which approximates the original random sampling with replacement by weighting instances according to a Poisson ($\lambda = 1$) distribution, and switching the λ to 6, it is possible to "leverage" resampling in order to increase the probability of assigning higher weights to instances while training the base model.

4.5 NAÏVE BAYES

Finally, **Naïve Bayes** is a classification technique based on Bayes' Theorem with an assumption of independence among predictors; this means that a Naïve Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

Bayes' Theorem provides a way of calculating posterior probability *P(A|B)* from *P(A)*, *P(B)* and *P(B|A)*, as:

$$P(A|B) = \frac{P(B|A)\ P(A)}{P(B)}$$

Here, *A* and *B* are two independent events, while *P(A)* and *P(B)* are their respective probabilities; *P(A|B)* and *P(B|A)* are the compound probabilities of *A* given *B* and *B* given *A*, respectively. We cannot obtain *P(A|B)* directly, however we can get *P(B|A)* and *P(A)* from the training data: B is our set of features in the dataset, while A is the set of all predictable classes. Theoretically, finding P(A|B) is not harder, however it becomes much more difficult as the number of features grows.

The advantages of using Naïve Bayes include:

- It is relatively simple to understand and build.

- It is easily trained, even with small datasets.
- It is extremely fast compared to other models.
- It is not sensitive to irrelevant features.

However, its main strength is also its weakness: the approach assumes that every feature is independent, which is not always the case.

5. *ONLINE DATASET ANALYSIS*

In order to use the dataset in online fashion, and simulate real-time learning, its entries are streamed from the arff file to MOA.

5.1 MOA

Three algorithms were used for this analysis: Adaptive Random Forest, Naïve Bayes and Hoeffding Tree. Amongst the three, the Adaptive Random Forest produces the best result, however it managed to do so with a significantly slower execution time.

Each of the three runs produced a set of parameters on which to evaluate the model; furthermore, every 100,000 processed instances, intermediary results were logged, in order to measure the growth of the predictor over time.

For each task run on the dataset, the following attributes have been measured:

- **Classified instances**: the number of entries in the dataset that has been processed so far by the classifier.
- **Time**: the execution time, in seconds, necessary for the classification.
- **Correct predictions (accuracy)**: percentage of entries successfully classified by the model so far.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Where, given a positive and a negative class we define:

TP – True Positives: the number of outcomes where the model correctly predicts the positive class.

TN – True Negatives: the number of outcomes where the model incorrectly predicts the positive class.

FP – False Positives: the number of outcomes where the model correctly predicts the negative class.

FN – False Negatives: the number of outcomes where the model incorrectly predicts the negative class.

- **Kappa score**: a statistic that is used to measure inter-rater reliability, a score of how much homogeneity exists in the classification.
- **Precision**: the ratio of correctly predicted positive observations to all positive observations in the positive class

$$\text{Precision} = \frac{TP}{TP + FP}$$

- **Recall**: the ratio of correctly predicted positive observations to all observations in the positive class

$$\text{Recall} = \frac{TP}{TP + FN}$$

- **F1-score (F-Measure)**: a weighted average of precision and recall. It takes into account both false positives and false negatives.

$$\text{F-Measure} = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

The following tables and graphs display the progression of the three algorithms during their execution:

Adaptive Random Forest

Classified instances	Accuracy	Kappa	Kappa temp.	Kappa M	Precision	Recall	F1-score	Time (s)
100,000	93.801	93.834	-58	93.8	0.721	0.715	0.718	347
200,000	98.205	98.245	43	98.2	0.809	0.792	0.801	732
300,000	93.809	93.823	-40	93.8	0.716	0.703	0.709	1135
400,000	91.202	91.215	-95	91.2	0.702	0.699	0.700	1578
500,000	94.395	94.391	-19	94.39	0.723	0.709	0.716	1966
581,012	98.476	98.452	20	98.4	0.821	0.786	0.805	2232

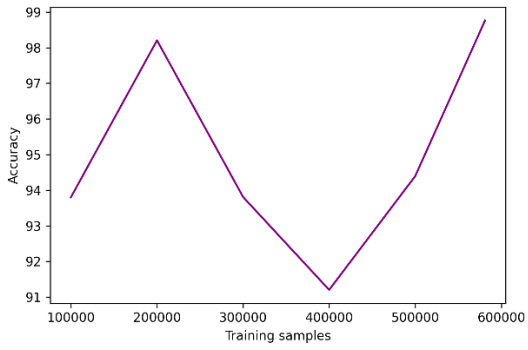


Figure 5.1 Adaptive Random Forest accuracy evolution.

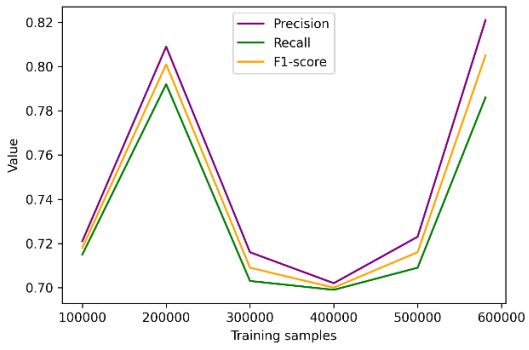


Figure 5.2 Adaptive Random Forest precision, recall and f1-score evolution.

This classifier is the one that produced the highest accuracy among the three, however its execution time was significantly slower compared to both Naïve Bayes and

Hoeffding Tree. Its accuracy reaches the lowest value at around 400,000 classified instances but manages to regrow by the end of the process.

Hoeffding Tree

Classified instances	Accuracy	Kappa	Kappa temp.	Kappa M	Precision	Recall	F1-score	Time (s)
100,000	90.901	90.907	-133	90.90	0.723	0.718	0.720	1.92
200,000	83.208	83.206	-424	83.20	0.642	0.606	0.623	3.45
300,000	75.207	75.202	-463	75.20	0.584	0.532	0.556	5.09
400,000	68.503	68.501	-599	68.50	0.503	0.498	0.500	6.82
500,000	75.605	75.603	-419	75.60	0.579	0.529	0.552	8.67
581,012	67.302	67.304	-1534	67.30	0.499	0.472	0.485	10.18

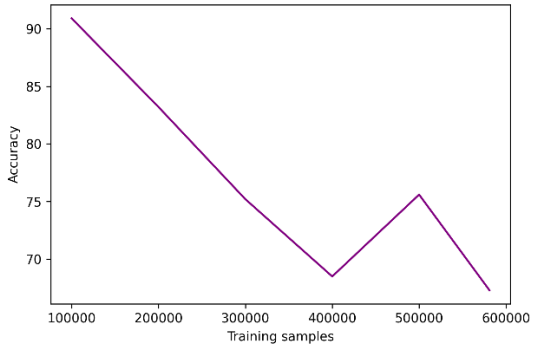


Figure 5.3 Hoeffding Tree accuracy evolution.

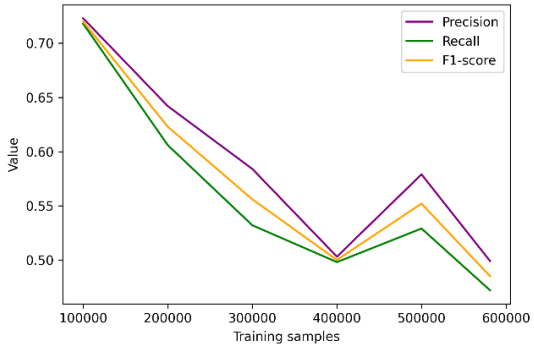


Figure 5.4 Hoeffding Tree precision, recall and f1-score evolution.

After the first 300,000 instances, we can see the accuracy oscillating. This suggests us that the classifier might have needed more samples before stabilizing its accuracy value.

Naïve Bayes

Classified instances	Accuracy	Kappa	Kappa temp.	Kappa M	Precision	Recall	F1-score	Time (s)
100,000	78.921	78.941	-441	78.90	0.589	0.472	0.524	1.75
200,000	52.691	52.672	-1381	52.60	0.392	0.314	0.348	3.46
300,000	64.341	64.312	-711	64.30	0.467	0.399	0.435	5.12
400,000	31.563	31.056	-1433	31.00	0.223	0.204	0.215	6.79
500,000	41.192	41.105	-1155	41.00	0.392	0.315	0.353	8.46
581,012	63.806	63.804	-1709	63.80	0.445	0.405	0.425	9.85

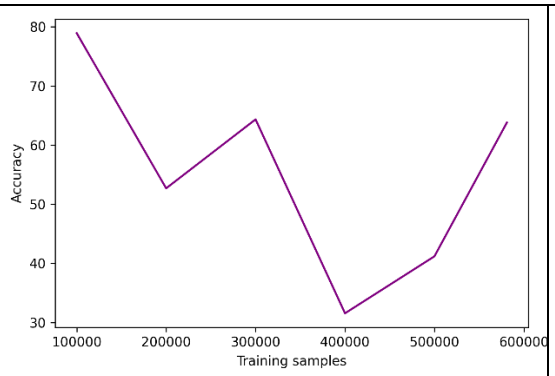


Figure 5.5 Naïve Bayes accuracy evolution.

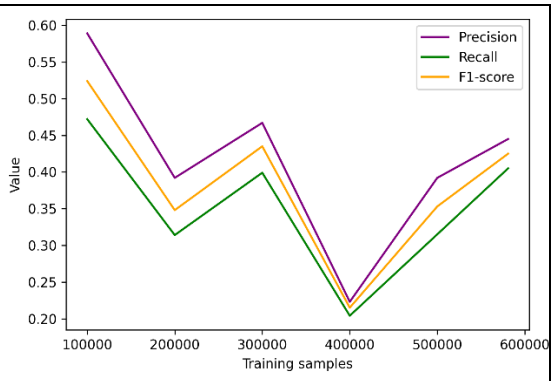


Figure 5.6 Naïve Bayes precision, recall and f1-score evolution.

This classifier performs a classic Bayesian prediction while making naïve assumption that all inputs are independent. It is a very simple algorithm with low computational costs, and this is reflected in its execution time, but also in its underwhelming accuracy when it comes to the considered problem.

By performing an early comparison between the three, we can easily notice how the Adaptive Random Forest algorithm clearly outclasses the other two in every observed metric. Therefore, if the extra training time is not an issue, the best choice easily points towards this tree classifier. Regarding the other two, they manage to stay relatively close, however the Naïve Bayes classifier seemed more "unstable" during the test runs: as we can see, after 400,000 classified instances the accuracy drops to 31%, while the Hoeffding Tree was much more consistent in its predictions.

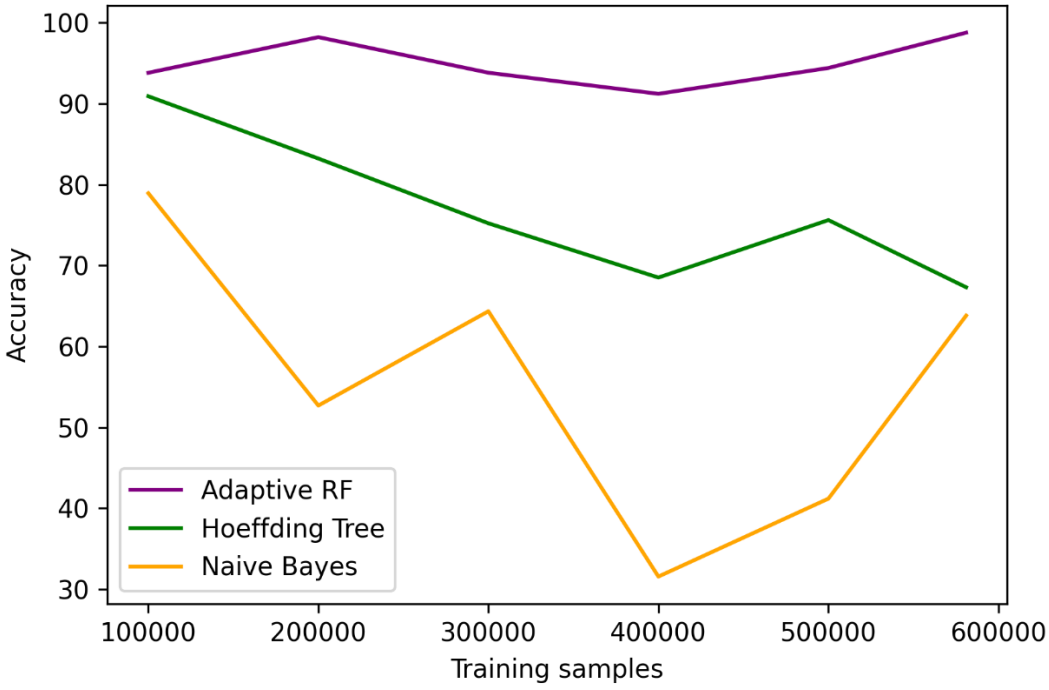


Figure 5.7 Accuracy comparison between the MOA classifiers.

5.2 *SCIKIT MULTIFLOW*

Just for the sake of comparison, we wanted to put MOA side-to-side with another powerful online machine learning library: Scikit Multiflow. Similarly to what was done in MOA, we took snapshots of the state of the algorithms during their execution. The results are reported ahead:

Adaptive Random Forest

#Instances	Accuracy	Kappa	Precision	Recall	F1-score	Time (s)
100,000	0.923	0.885	0.785	0.753	0.769	1221
200,000	0.931	0.889	0.790	0.754	0.772	2463
300,000	0.931	0.893	0.786	0.752	0.769	3743
400,000	0.933	0.895	0.799	0.755	0.777	5085
500,000	0.936	0.902	0.793	0.755	0.776	6291
581,012	0.940	0.902	0.798	0.760	0.777	7386

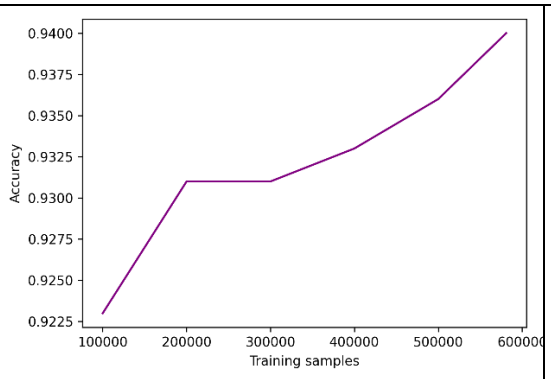


Figure 5.8 Adaptive Random Forest accuracy evolution.

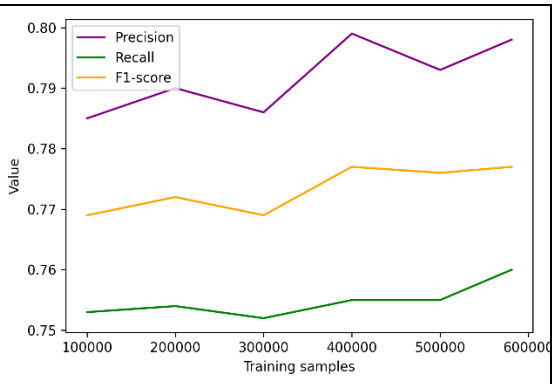


Figure 5.9 Adaptive Random Forest precision, recall and f1-score evolution.

Hoeffding Tree

#Instances	Accuracy	Kappa	Precision	Recall	F1-score	Time (s)
100,000	0.837	0.657	0.658	0.587	0.615	257
200,000	0.848	0.680	0.646	0.612	0.627	398
300,000	0.836	0.720	0.643	0.635	0.635	586
400,000	0.831	0.720	0.645	0.636	0.637	770
500,000	0.826	0.717	0.640	0.644	0.639	940
581,012	0.820	0.710	0.640	0.648	0.624	1122

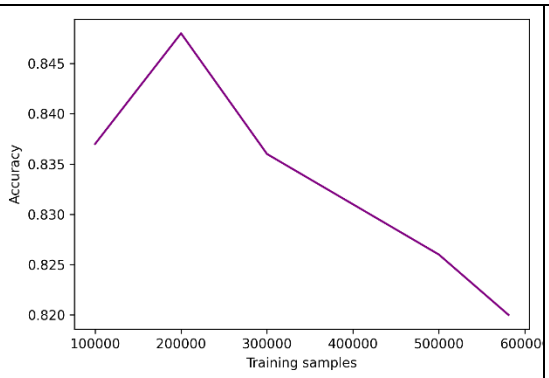


Figure 5.10 Hoeffding Tree accuracy evolution.

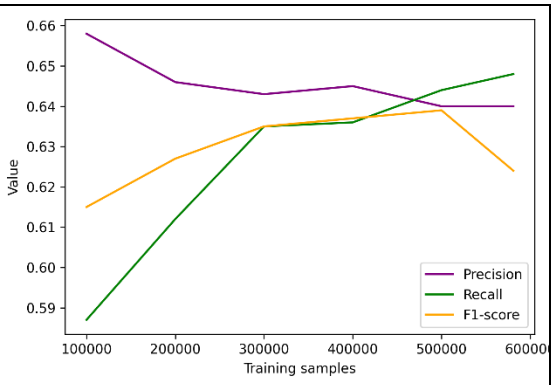


Figure 5.11 Hoeffding Tree precision, recall and f1-score evolution.

Naïve Bayes

#Instances	Accuracy	Kappa	Precision	Recall	F1-score	Time (s)
100,000	0.725	0.314	0.550	0.434	0.457	284
200,000	0.692	0.216	0.515	0.458	0.420	504
300,000	0.631	0.309	0.467	0.416	0.398	723
400,000	0.620	0.324	0.426	0.355	0.354	943
500,000	0.589	0.280	0.407	0.318	0.321	1160
581,012	0.590	0.301	0.422	0.319	0.327	1408

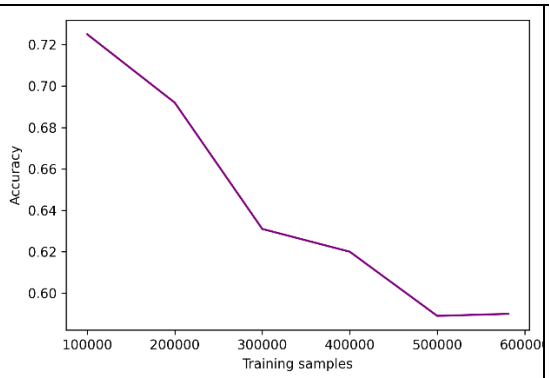


Figure 5.12 Naïve Bayes accuracy evolution.

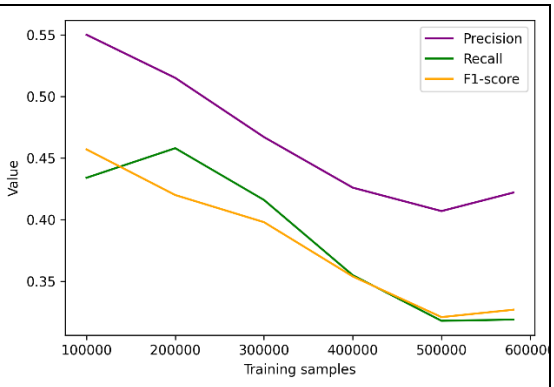


Figure 5.13 Naïve Bayes precision, recall and f1-score evolution.

After a quick first analysis we can easily notice how similar most of the results are;

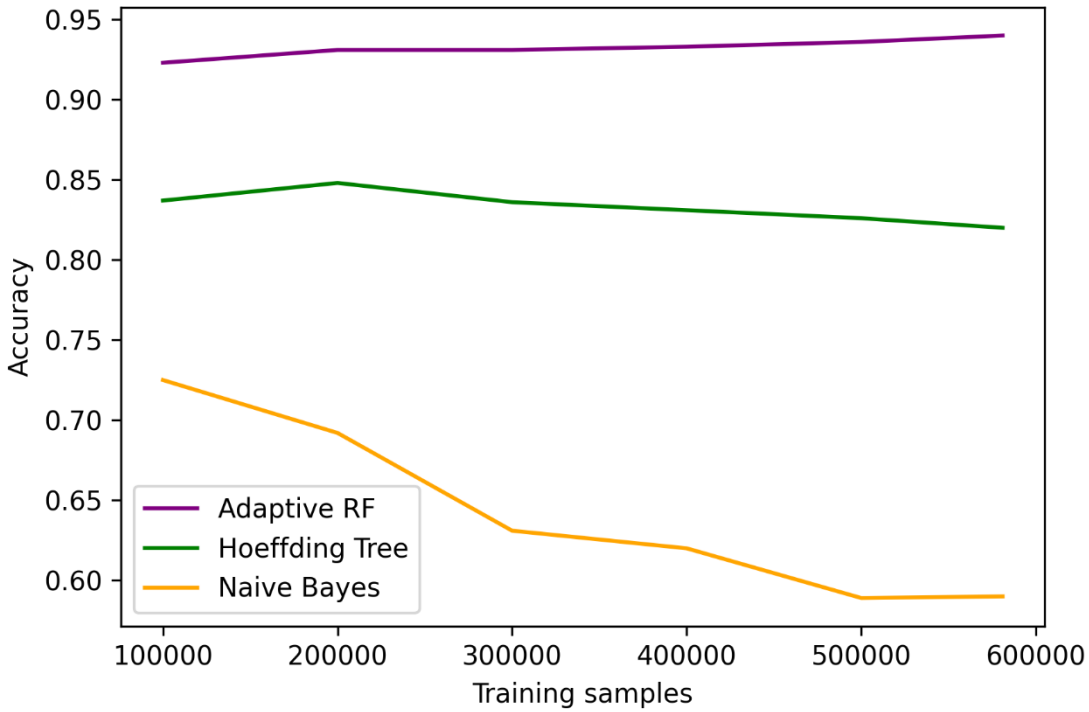


Figure 5.14 Accuracy comparison between the Scikit Multiflow classifiers.

6. BATCH LEARNING APPROACH

To further expand on the analysis of the goodness of the results obtained in MOA and Scikit Multiflow, we decided to compare these findings with other analysis techniques using batch learning. In particular, the focus shifted towards one of the most widely used libraries for machine learning, python's scikit learn.

In this section we will examine the results obtained by employing different batch machine learning solutions. Furthermore, we will discuss on the performances of splitting the dataset vs. employing k-cross fold validation for the evaluation of the performances and we will analyse overfitting and underfitting in our data.

6.1 DATASET SPLITTING VERSUS CROSS VALIDATION

Learning the parameters of a prediction function and testing it on the same data is a methodological mistake: a model that would just repeat the labels of the samples that it has just seen would have a perfect score but would fail to predict anything useful on unseen data. This condition is called overfitting, as we will see in the next section. To avoid this, it is common practice when performing a supervised machine learning task to hold out part of the available data as test data. When evaluating different settings for estimators (their hyperparameters) there is still a risk of overfitting on the test set because the parameters can be tweaked until the estimator performs optimally. This way knowledge about the data can "leak" into the model and evaluation metrics no longer report on generalization performance. To solve this problem, yet another part of the dataset can be held as a "validation set": training proceeds on the train set, after which evaluation is done on the validation set, and when the experiment seems to be successful, the final evaluation on the test set is performed. However, by partitioning the available data into three sets, we drastically reduce the number of samples which can be used for learning the model, and the results can depend on a particular random choice for the pair of sets.

A solution to this problem is a procedure called cross-validation (CV). A test set should still be held out for final evaluation, but the validation set is no longer required when performing CV. In the basic approach, called k-fold CV, the training set is split into k smaller sets; then, for each of the k "folds", the next procedure in followed:

- A model is trained using k-1 of the folds as training data.
- The resulting model is validated on the remaining part of the data.

Finally, the performance measure reported by the k-fold cross-validation is the average of the values computed in the loop. This approach can be computationally expensive, but does not waste as much data, which is a major advantage in most problems.

The following tables and graphs report the improvement obtained with 10-fold cross-validation:

Random Forest

Train-test split:

Accuracy	Precision	Recall	F1-Score	Kappa	Time(s)
0.931	0.943	0.868	0.899	0.888	36.8

10-fold cross validation:

	Accuracy	Precision	Recall	F1-Score	Kappa
Fold 1	0.928	0.940	**0.867**	0.897	0.883
Fold 2	0.927	0.942	0.863	0.895	0.881
Fold 3	0.926	0.940	0.856	0.890	0.880
Fold 4	0.927	0.943	0.864	0.896	0.882
Fold 5	0.927	**0.946**	0.855	0.892	0.883
Fold 6	0.929	0.940	0866	0.895	0.885
Fold 7	0.928	0.945	0.863	0.895	0.883
Fold 8	0.928	0.944	0.866	**0.898**	0.886
Fold 9	0.927	0.935	0.862	0.890	0.882
Fold 10	**0.931**	0.940	0.866	0.896	**0.888**

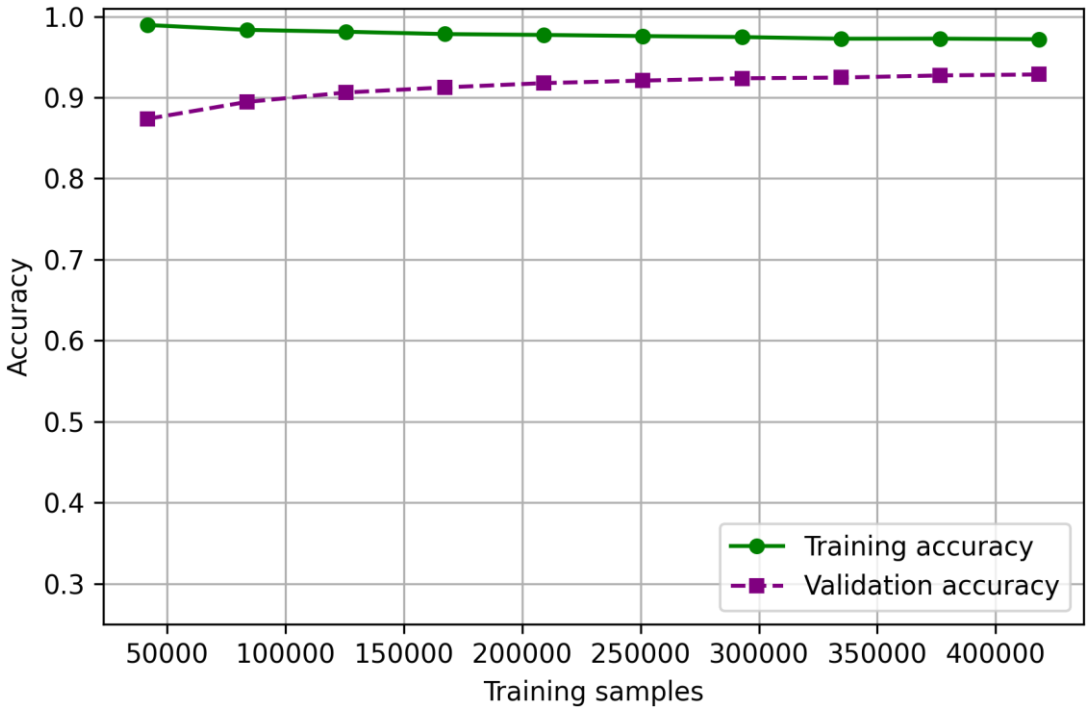


Figure 6.1 Mean accuracy of the 10-fold CV for the Random Forest classifier.

Decision Tree

Train-test split:

Accuracy	Precision	Recall	F1-Score	Kappa	Time(s)
0.921	0.898	0.849	0.870	0.873	5.36

10-fold cross validation:

	Accuracy	Precision	Recall	F1-Score	Kappa
Fold 1	0.917	0.887	**0.863**	**0.874**	0.866
Fold 2	0.916	**0.890**	0.849	0.867	0.865
Fold 3	0.918	0.888	0.840	0.860	0.868
Fold 4	0.918	0.882	0.846	0.862	0.868
Fold 5	0.916	0.885	0.845	0.863	0.866
Fold 6	0.916	0.878	0.857	0.867	0.865
Fold 7	0.918	0.887	0.860	0.871	0.868
Fold 8	0.919	0.883	0.859	0.870	0.870
Fold 9	0.915	0.884	0.845	0.861	0.863
Fold 10	**0.921**	0.887	0.850	0.867	**0.872**

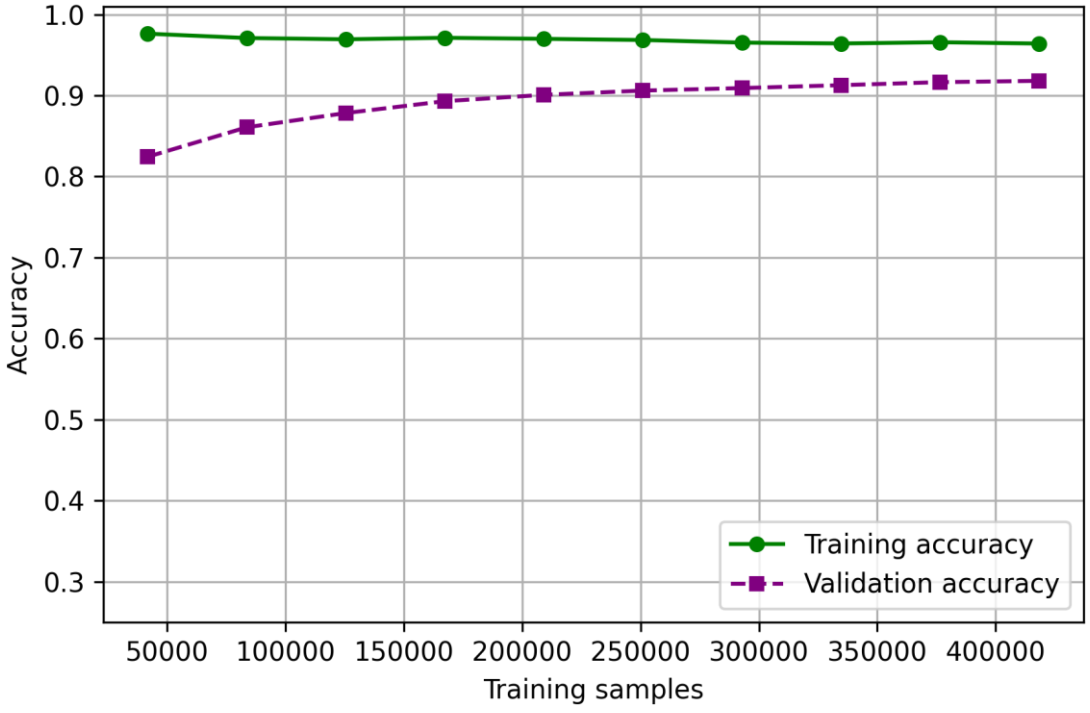


Figure 6.2 Mean accuracy of the 10-fold CV for the Decision Tree classifier.

Naïve Bayes

Train-test split:

Accuracy	Precision	Recall	F1-Score	Kappa	Time(s)
0.453	0.397	0.594	0.367	0.258	0.866

10-fold cross validation:

	Accuracy	Precision	Recall	F1-Score	Kappa
Fold 1	0.457	0.392	0.591	0.363	0.260
Fold 2	0.454	0.395	0.595	0.361	0.256
Fold 3	0.457	0.389	0.588	**0.364**	**0.261**
Fold 4	0.457	0.389	**0.592**	0.361	0.258
Fold 5	**0.457**	**0.395**	0.587	0.361	0.260
Fold 6	0.456	0.394	0.584	0.363	0.260
Fold 7	0.455	0.387	0.588	0.357	0.258
Fold 8	0.451	0.389	0.577	0.353	0.256
Fold 9	0.456	0.395	0.582	0.359	0.259
Fold 10	0.455	0.389	0.584	0.361	0.259

Figure 6.3 Mean accuracy of the 10-fold CV for the Naïve Bayes classifier.

6.2 *OVERFITTING AND UNDERFITTING EVALUATION*

Two particularly important aspects worth considering when evaluating the performance of batch classifiers are over- and underfitting. Overfitting refers to the condition by which a model learns too much from the training dataset, thus resulting in poor generalization capabilities. This can happen if the noise or random fluctuations in the training dataset are picked up and learned as concepts by the model; the problem is that these concepts do not apply to new data. Overfitting is more likely with non-parametric and non-linear models that have more flexibility when learning a target function. As such, many non-parametric machine learning algorithms also include parameters or techniques to limit and constrain how much detail the model learns.

As an example, decision trees and random forests are two kinds of machine learning algorithms that are very flexible and thus are often subject to overfitting on data. This problem can be addressed by pruning a tree after it has learned in order to remove some of the details it has picked up.

Underfitting, on the other hand, refers to a model that can neither fit training data nor generalize to new, unseen, data. An underfit machine learning model is not a suitable model and will be obvious as it will have poor performances on the training data. Underfitting is often not discussed as it easily detectable given a good performance metric. The remedy is to move on try alternate machine learning algorithms; nevertheless, it does provide a good contrast to the problem overfitting.

When tuning the hyperparameters of our two tree classifiers we settled for a maximum depth of … and … for the Random Forest and Decision Tree specifically. Why such a small number? Simply, by using a higher depth for the trees, we would quickly encounter an overfit model. The following two graphs highlight the accuracy curves of the two classifiers when the max depth of the trees is left untouched:

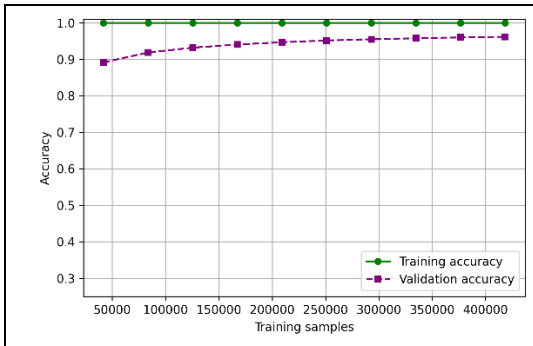


Figure 6.4 Overfitting in the Random Forest classifier.

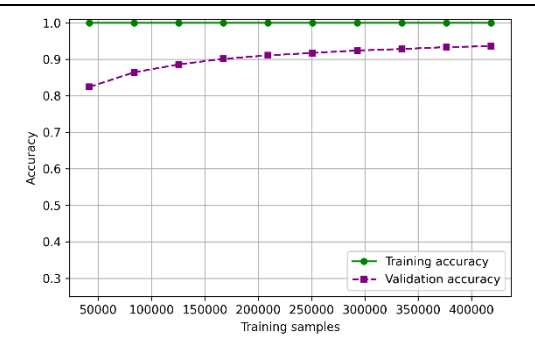


Figure 6.5 Overfitting in the Decision Tree classifier.

We can clearly see that, while the validation accuracy increases as it should, the training accuracy remains constant from the start. So, why does this happen? By default, the trees in these classifiers are not pruned; thus, roughly around *(1 – 1/e)* of the trees will contain the training case and will be able to lookup the correct solution, resulting in a higher apparent accuracy. Even if the *1/e* trees which were not trained with this case would always predict wrongly, they would always be outvoted by the others.

6.3 *CONFUSION MATRIX*

A confusion matrix is a table layout that allows the visualization of the performance of the employed classifiers. Each row of the matrix represents the instances in the actual class, while each column represents the instances in a predicted class, or vice versa.

Random Forest

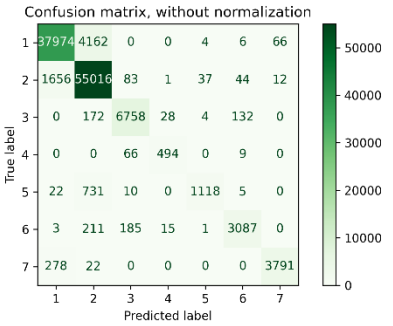


Figure 6.6 Random Forest confusion matrix,

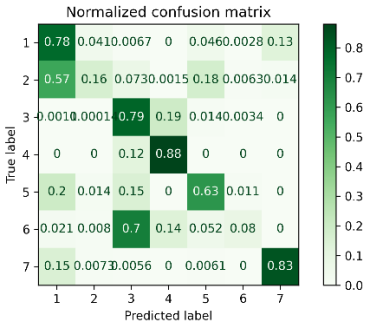


Figure 6.7 Normalized Random Forest confusion matrix,

Decision Tree

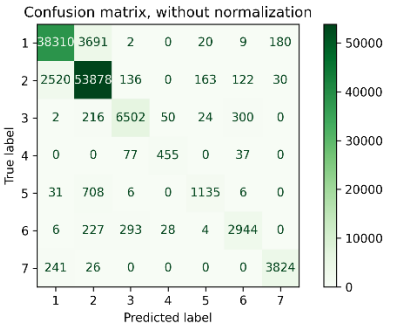


Figure 6.8 Decision Tree confusion matrix.

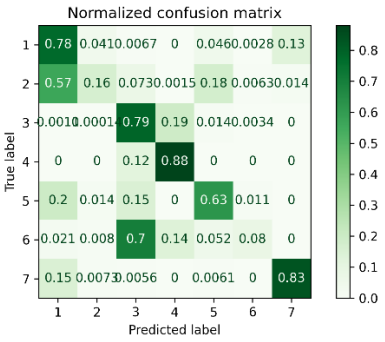


Figure 6.9 Normalized Decision Tree confusion matrix.

Naïve Bayes

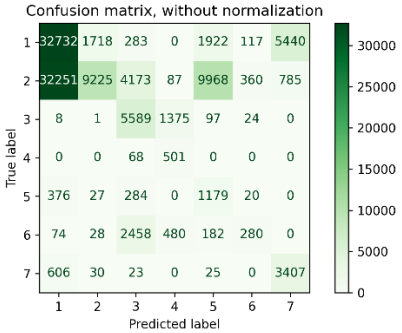


Figure 6.10 Naïve Bayes confusion matrix.

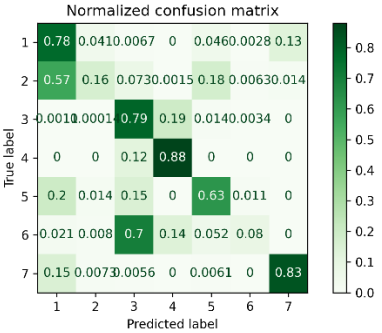


Figure 6.11 Normalized Naïve Bayes confusion matrix.

7. *CONCLUSIONS*

This last table provides the final comparison between all the different employed techniques and classifiers (the batch learning results are the best obtained across the various steps previously discussed):

	Accuracy	Precision	Recall	F1-score	Kappa	Time(s)
MOA						
Adaptive Random Forest	0.98				0.98	2400
Naïve Bayes	0.63				0.63	9.39
Hoeffding Tree	0.78				0.78	9.25
Scikit multiflow						
Adaptive Random Forest	0.94	0.79	0.76	0.77	0.90	7386
Naïve Bayes	0.57	0.43	0.36	0.36	0.27	1408
Hoeffding Tree	0.82	0.64	0.62	0.63	0.71	1122
Scikit learn						
Random Forest	0.99	0.99	0.99	0.99	0.99	163
Naïve Bayes	0.99	0.99	0.99	0.99	0.99	2.77
Decision Tree	0.99	0.99	0.99	0.99	0.99	3.06

Since most learners use a greedy, hill-climbing search in the space of models, they are much more prone to overfitting, local maxima, etc.

This may be deceiving

Given the nature of online learning algorithms, its … they overfit… that's why the online measurements are overall better..

REFERENCES

[1] Forest Cover Type Classification Study, Thomas Kolasa and Aravind Kolumum Raaja, Amazon AWS, 2016.

[2] Adaptive random forests for evolving data stream classification, Heitor M. Gomes et al., 2017.

[3] A new ensemble approach for dealing with concept drift, Leandro L. et al., IEEE Transactions on Knowledge and Data Engineering, 2012.