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: 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 of 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.

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

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]

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.

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.

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; the following graph highlights their distribution in the seven cover type classes:

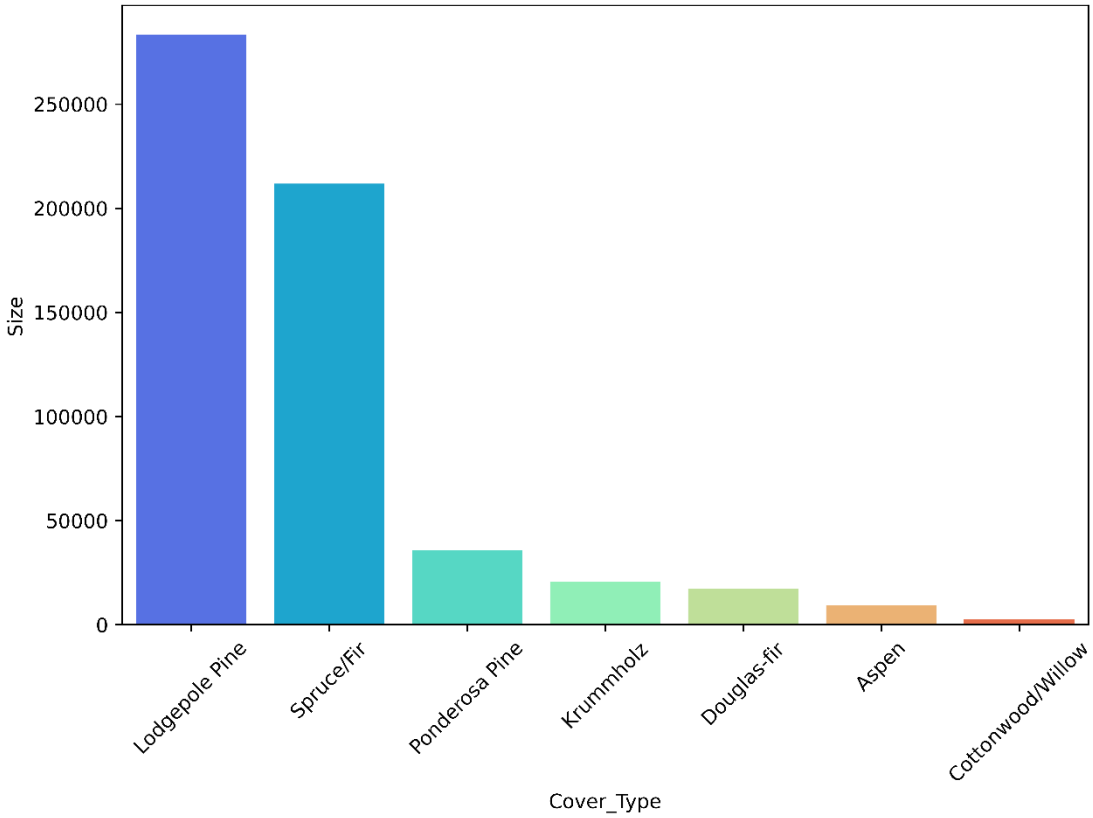


Figure 3.1 Dataset distribution according to the target attribute.

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.

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”.

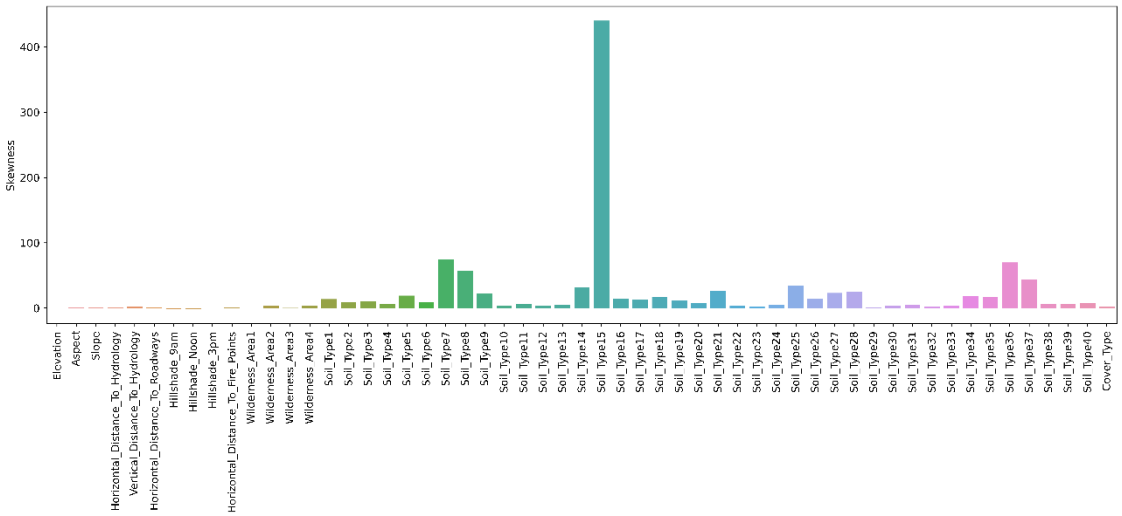
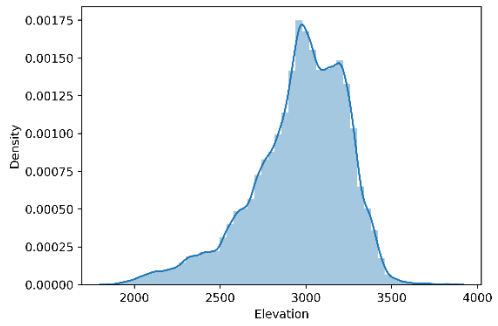


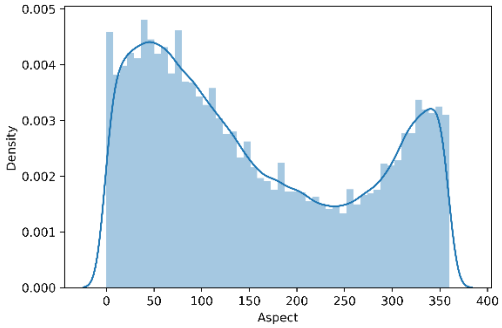
Figure 3.2 Skewness of each feature in the model.

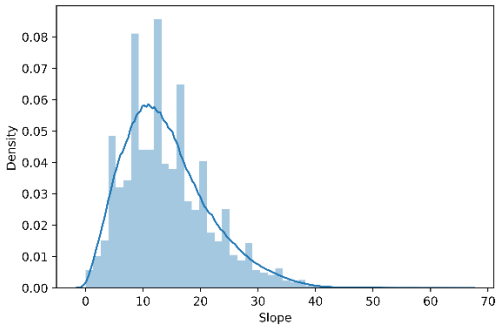
Given the different nature of some of the features in the dataset (i.e., continuous, and binary features), it is worth splitting the dataset into four categories, in order to check the number of value counts within each feature; the considered subsets are:

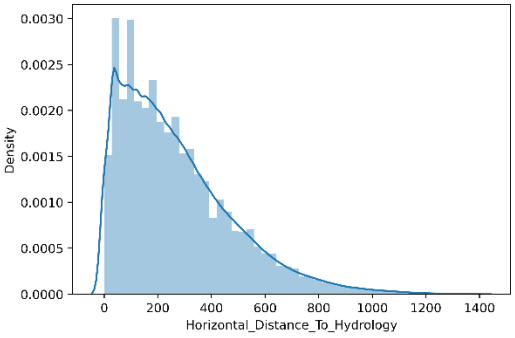
- Continuous data: this includes elevation, aspect, slope, horizontal and vertical distance to hydrology, horizontal distance to roadways, the three hill shades and the horizontal distance to fire points.
- Binary data: the complete set of binary features, including the 4 wilderness areas and the 40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

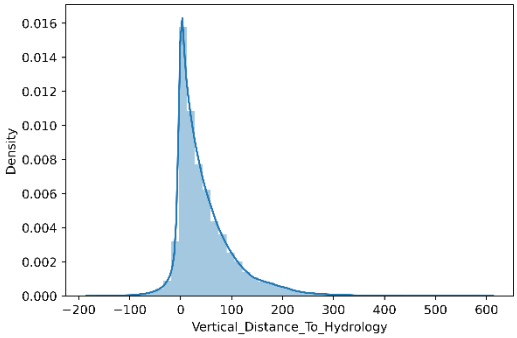
This first set of graphs highlights the distribution of the continuous attributes of the dataset:

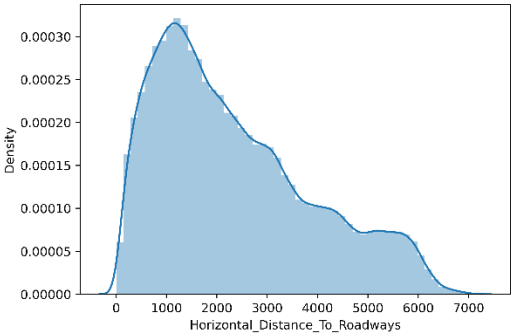


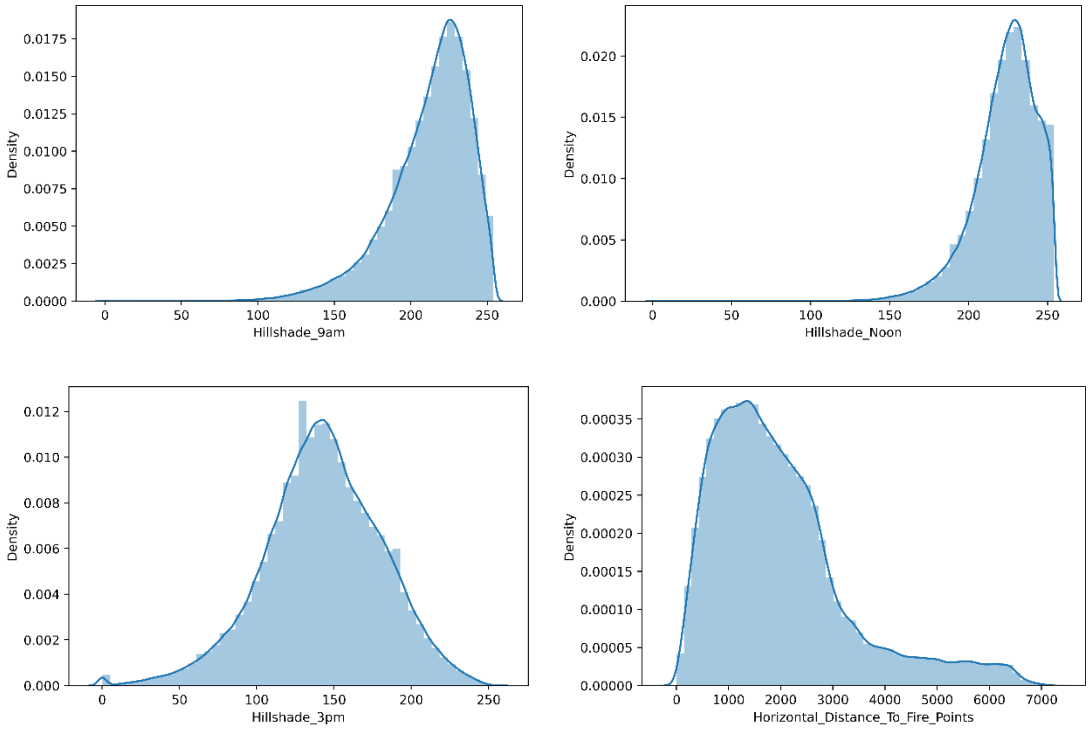












Another interesting feature to analyse are the wilderness areas of the trees in the dataset, in relation to each cover type:

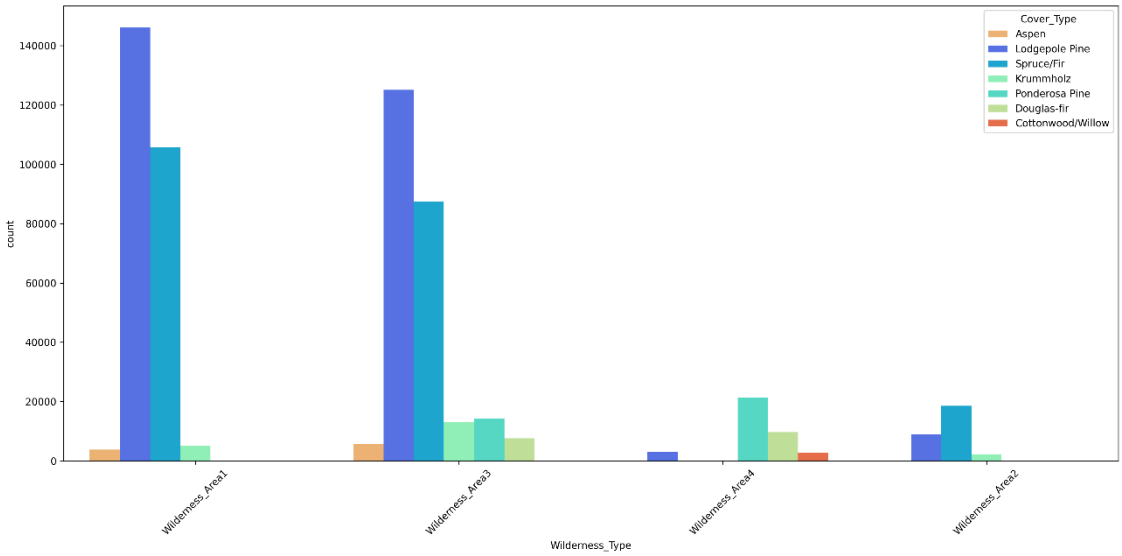


Figure 3.3 Distribution of trees among the four wilderness areas considered.

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:

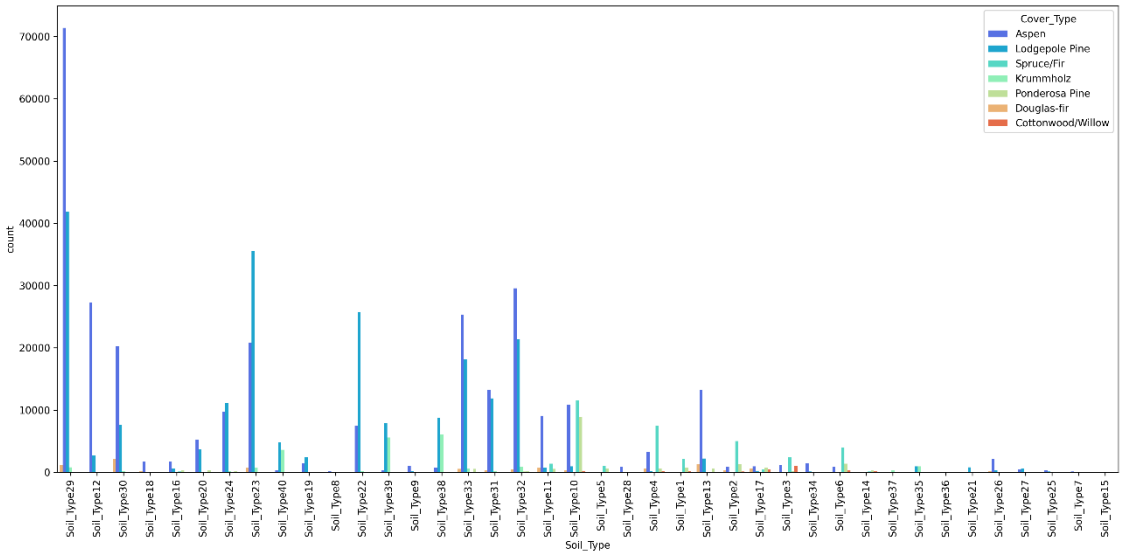


Figure 3.4 Distribution of soil types.

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.

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.

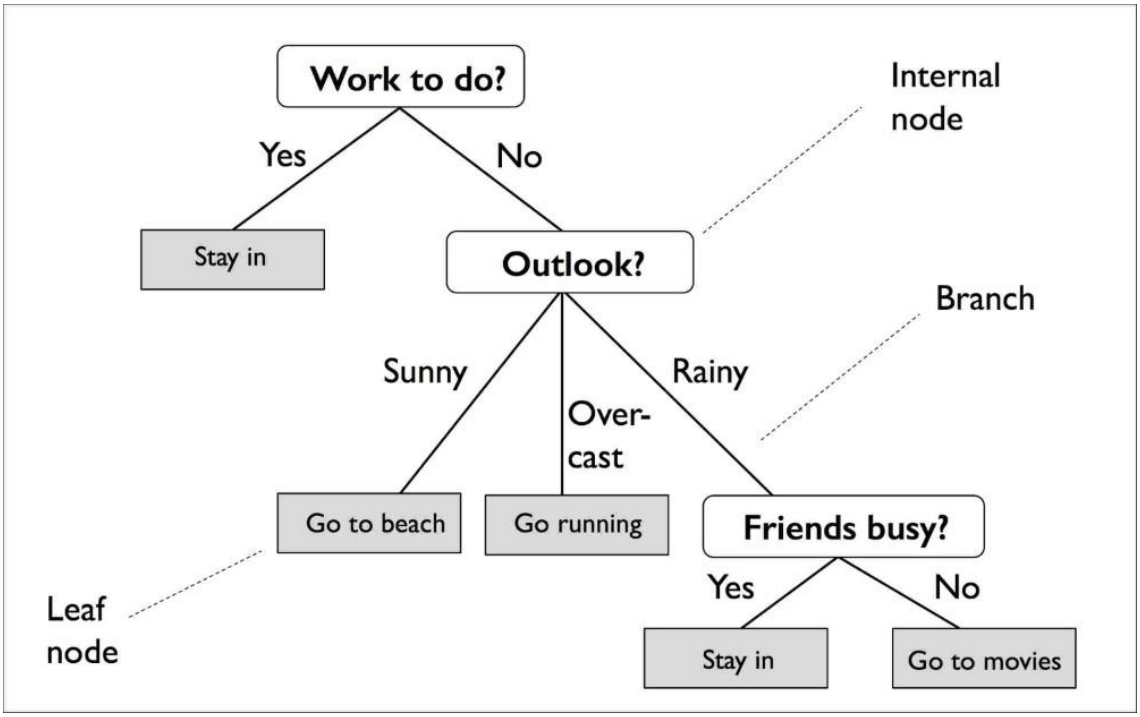


Figure 4.1 Decision Tree example.

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.

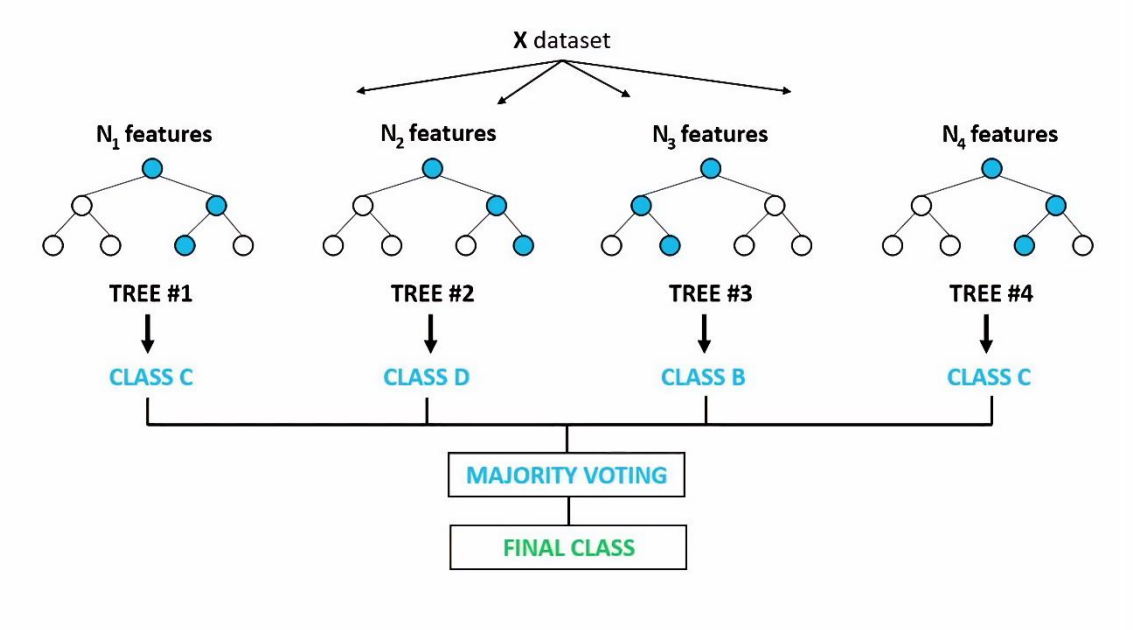


Figure 4.2 Basic process behind the Random Forest algorithm.

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.

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.

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. Three classifiers were used for this analysis: adaptive random forest, naïve bayes and hoeffding tree. Among these three, the random forest produces the best result, however its execution time is significantly slower.

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 classifier 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: percentage of entries successfully classified by the model so far.
- 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
- Recall: the ratio of correctly predicted positive observations to all observations in the positive class
- F1-score: a weighted average of precision and recall. It takes into account both false positives and false negatives.

The following tables display the progression of the three classifiers during their execution:

Adaptive Random Forest

Classified instances	Time (s)	Correct pred(%)	Kappa	Kappa temp.	Kappa M	Precision	Recall	F1
100,000	347	93.80	93.8	-58	93.8			
200,000	732	98.20	98.2	43	98.2			
300,000	1135	93.80	93.8	-40	93.8			
400,000	1578	91.20	91.2	-95	91.2			
500,000	1966	94.39	94.39	-19	94.39			
581,012	2232	98.4	98.4	20	98.4			

The Adaptive Random Forest classifier is an adaptation of the original Random Forest algorithm which has been successfully applied to various machine learning tasks. The "Adaptive" part comes from its mechanisms to adapt to different kinds of concept drifts, given the same hyper-parameters. 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.

Naïve Bayes

Classified instances	Time (s)	Correct pred(%)	Kappa	Kappa temp.	Kappa M	Precision	Recall	F1
100,000	1.75	78.90	78.90	-441	78.90			
200,000	3.46	52.60	52.60	-1381	52.60			
300,000	5.12	64.30	64.30	-711	64.30			
400,000	6.79	31.00	31.00	-1433	31.00			
500,000	8.46	41.00	41.00	-1155	41.00			
581,012	9.85	63.80	63.80	-1709	63.80			

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.

Hoeffding Tree

Classified instances	Time (s)	Correct pred(%)	Kappa	Kappa temp.	Kappa M	Precision	Recall	F1
100,000	1.92	90.90	90.90	-133	90.90			
200,000	3.45	83.20	83.20	-424	83.20			
300,000	5.09	75.20	75.20	-463	75.20			
400,000	6.82	68.50	68.50	-599	68.50			
500,000	8.67	75.60	75.60	-419	75.60			
581,012	10.18	67.30	67.30	-1534	67.30			

A Hoeffding tree is an incremental decision tree induction algorithm that is capable of learning from massive data streams, assuming that the distribution of the data does not change significantly over time. 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.

6. *COMPARISON WITH OTHER TECHNIQUES*

To further expand on the analysis of the goodness of the results obtained in MOA, we decided to compare these findings with other analysis techniques, using both online and batch learning. In particular, the focus shifted towards one of the most widely used libraries for machine learning, python's scikit learn, and its online counterpart, scikit multiflow. First of all, a direct online comparison is due: the scikit multiflow library allows us to design and run experiments by making use of powerful stream learning methods and evaluators. To better compare the results, the same three classifiers have been used.

The next table summarizes the final measurements obtained with all of the employed techniques:

	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

7. CONCLUSIONS

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.