Decision Tree Learning

Proseminar Data Mining

Philipp Hagenlocher
Fakultät für Informatik
Technische Universität München
Email: philipp.hagenlocher@in.tum.de

***Abstract*— Decision tree learning is a concept that describes a process for building decision trees from data in order to do classification and regression tasks. In this paper the induction of such decision trees, for classification tasks, will be discussed with emphasize on the algorithms *ID3* and *C4.5*. Other topics such as evaluation, pruning and ensemble learning, with random forests, will be mentioned and briefly summarized.**

***Index Terms*— ID3, C4.5, Induction, Classification**

I. What are decision trees?

This chapter will explain the motivation for classification and regression in data mining and introduce what decision trees are. It will also discuss on how decision trees are used to do these tasks.

A. The motivation

As described by Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth [1] data mining is a part of knowledge discovery in databases. In order to extract knowledge from data 3 main techniques are used:

- Classification
  - The task of splitting up the data in predefined classes and classifying new data
- Regression
  - The task of producing a prediction for future data based on past data
- Clustering
  - The task of finding possible categorizations of the data

Classification and regression are so called *supervised learning methods* since one have to provide learning data for ones algorithms. Clustering on the other hand does not use any learning data and tries to find classes to put the data in. Decision trees can be used for all of these tasks [2, pg.85-89] but this paper will only discuss classification and regression trees.

B. General information

A decision tree, in a mathematical sense, is a acyclic graph with a fixed root. A node describes an attribute in the data and the edges describe a decision based on this attribute. Even though most examples use binary decisions it is important to note that a node can have as many edges as they want. In operations research these trees are understood as decisions and consequences [2, pg.10]. In general the tree is traversed from the root, using the attribute in the node to choose a new node. Depending on the task, the leaf has a different meaning. This will be discussed now.

C. Usage of decision trees

1) Classification: The provided example in figure 1 is a simple decision tree to do a classification for an insurance company. Let a new potential customer be 30 years old and has medical problems. That can be used to classify the risk for that customer. We will start at the root and go down the *No* edge since our customer is younger than 55 years. After that, we go down the *Yes* edge since our customer does have medical problems. The leaf node is the class that our new data point belongs to. Thus it is obvious that the new potential customer brings high risk with them.

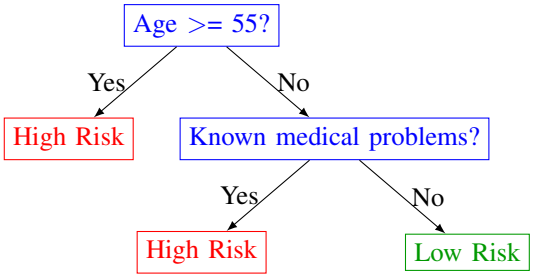


Fig. 1: Simple insurance risk classification

2) Regression: Regression itself is a very different task from classification although it becomes very similar in the context of decision trees. Instead of classifying new information the tree (in this case called *regression tree*) it has to give a prediction based on given data. This can be achieved by using linear regression models as the leafs in the trees as demonstrated by Rokach and Maimon [2, pg.163]. There are other methods such as taking the mean of the samples as described by Loh [3, pg.18]. Thus, regression with decision trees is very similar to classification since the trees are taking data and classifying it to a regression model or even precomputed value. The important difference is that the target is not discrete, but a continuous value. This value would most likely be the mean of all the targeted values like the CHAID and CART algorithms do.

II. Induction of decision trees

In this chapter the induction of decision trees for classification tasks will be demonstrated by looking at the ID3 and the C4.5 algorithm. Other algorithms will be discussed in a very short manner.

A. A general approach

When creating a tree there has to be training set supplied. This training set has to be labeled so this task is a supervised learning task. In general the induction task itself is quite easy and can be summarized as follows:

1) If a *stopping criterion* has been reached return the finished tree
2) Otherwise use a *splitting criterion* in order to partition the training set
3) For every partition in the training set start recursively with 1.) again

B. Goals

When creating a decision tree, the complexity (number of nodes) should be high enough to correctly classify new data but shouldn't be too high to retain comprehensibility and to avoid overfitting, which will be discussed in chapter III. In general the goal is to have an induction algorithm (also called an *inducer*) to split the data into finite subsets with every new node based on a splitting criterion [2, pg.29]. The tree should remain as simple as possible [4, pg.87]. Creating all possible decision trees and then choosing the simplest is not viable since the computation will become too intensive. Instead an inducer is used to generate a tree which is simple enough. Such an inducer will be discussed in the next section.

C. ID3

The ID3 algorithm was created in order to have a simple enough tree without much computation. This is why the algorithm does not use any pruning, which will be discussed in cahpter IV-A, and does not reuse an attribute of the training data twice. The algorithm was published by Quinlan [4] and will be introduced in a way to make it more sensible for a software implementation.

1) General overview: In the following notation the training set T will be defined as a set of tuples with attributes and their corresponding values. For a training set T the partition on attribute a will be defined as T_a. The algorithm can be defined as algorithm 1.

When the training set is partitioned on an attribute, a set of new training sets is returned. Each will only contain the same value for this attribute. A generic algorithm for the partitioning is defined as algorithm 2. The *stopping criterion* for ID3 can be described as: *Create a leaf once the data cannot be split any further.*. In order to understand the *splitting criterion* it is required to understand what information gain is.

Algorithm 1 ID3

procedure ID3(T)
  if T only consists of the same target values **then**
    return Leaf with target value
  else if T has the same values for the same attributes but different target values **then**
    return Leaf with target value that occurs most
  end if
  $a \leftarrow$ Attribute in T with highest information gain
  $node \leftarrow$ Node with attribute a
  $partitions \leftarrow$ PARTITION(T,a)
  for $partition$ in $partitions$ **do**
    $subtree \leftarrow ID3(partition)$
    Add $subtree$ to $node$
  end for
  return $node$
end procedure

Algorithm 2 Partitioning

procedure PARTITION(T, a)
  $partitions \leftarrow \emptyset$
  for every value v that the attribute a can have in T **do**
    $partition \leftarrow$ all sets in T that have the value v on attribute a
    Add $partition$ to $partitions$
  end for
  return $partitions$
end procedure

2) Information entropy and information gain: The algorithm uses the definition of entropy by Shannon. For a random variable **X** and a probability function for the random variable **P** the self contained information is defined as:

$$I(\mathbf{X}) = -log_2\mathbf{P}(\mathbf{X})$$

The self contained information is measured in bits. An easy example is a fair coin toss. Since there are two states that have the same probability, the contained information should be a single bit:

$$I(head) = -log_2\mathbf{P}(head) = -log_2 0.5 = 1bit$$

Based on this definition, the entropy of a random variable is defined as:

$$H(\mathbf{X}) = \sum_{x\in\mathbf{X}} \mathbf{P}(x)I(x) = -\sum_{x\in\mathbf{X}} \mathbf{P}(x)log_2\mathbf{P}(x)$$

The entropy on a training set introduced in this chapter has to be calculated slightly differently since it is not a random variable. Instead of a real probability the *empirical probability* of the different targets in the training data is used. Using the notation from before, the entropy on a training set T partioned by the $target$ is defined as:

$$H(T) = -\sum_{tp\in T_{Target}} \frac{|tp|}{|T|} log_2 \frac{|tp|}{|T|}$$

TABLE I: Training examples

Name	Can fly?	Can swim?	>=4 Legs?	Targeted class
Sparrow	Yes	No	No	Air
Parrot	Yes	No	No	Air
Kingfisher	Yes	Partially	No	Air
Eagle	Yes	No	No	Air
Shark	No	Yes	No	Sea
Dolphin	No	Yes	No	Sea
Whale	No	Yes	No	Sea
Bear	No	Yes	Yes	Land
Dog	No	Yes	Yes	Land
Cat	No	Partially	Yes	Land

Based on this definition, the information gain is defined as:

$$IG(T,a) = H(T) - \sum_{tp \in T_a} \frac{|tp|}{|T|} * H(tp)$$

Information gain can be understood as a change of entropy, if T gets partitioned by a. Quinlan describes maximising the information gain as the minimization of the mutual information between the attribute a and the set T [4, pg.90 (footnote)]. So by using the information gain as a splitting criterion, the algorithm tries to use the attribute, that will contain the most information on splitting the data, first.

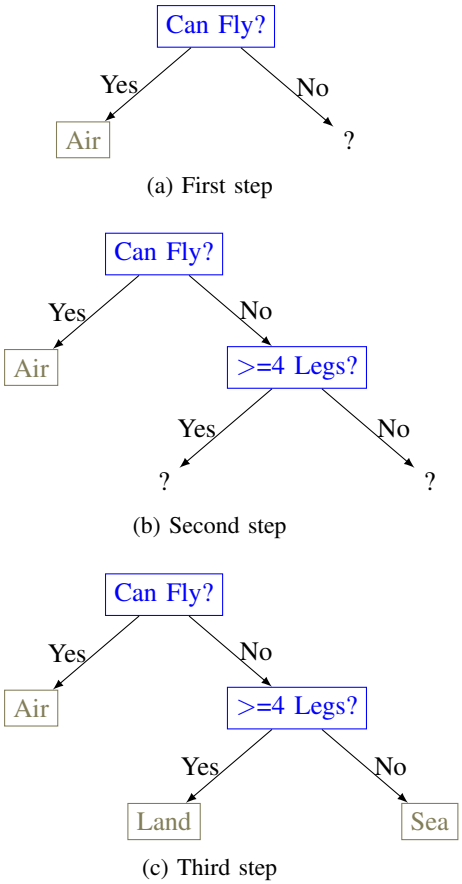
3) An example for ID3: As an example we will create a classification tree using ID3 for classifying animals into *sea*, *land* and *air* animals. The training examples are defined in table I.

In order to start with the algorithm we will try to find the attribute with the most information gain in order to make our first node in the tree and split our data. Of course the attribute *Name* is just an identifier and will not be considered when building the tree. Calculating the information gain for the attribute *Can Fly?* looks like this:

$$\begin{aligned} IG(T,\text{"}CanFly?\text{"}) &= H(T) - \sum_{tp \in T_{\text{"}CanFly?\text{"}}} \frac{|tp|}{|T|} * H(tp) \\ &\approx 1.57 - \sum_{tp \in T_{\text{"}CanFly?\text{"}}} \frac{|tp|}{|T|} * H(tp) \\ &= 1.57 - (\frac{4}{10} * 0 + \frac{6}{10} * 1) = 0.97 \end{aligned}$$

In this case *Can swim?* and *>=4 Legs?* have an information gain of around 0.88 so *Can Fly?* will be the attribute for the first node. Splitting the training set on the different values of the attribute (in this case it is only *Yes* and *No*) reveals, that the data left on the *Yes* side has the same target class for every data point. Thus a node for the class *Air* is added to the root as it is shown in figure 2a.

After this split the information gain for *Can swim?* and *>=4 Legs?* are 0.19 and 1.0 respectively so the next split is obviously *>=4 Legs?*. Looking at the newly partitioned data reveals, that the new sets for the leafs only consist of the same targets again. Therefore the algorithm adds the new leafs and finishes the process as seen on figures 2b and 2c.



(a) First step

(b) Second step

(c) Third step

Fig. 2: Full example

4) Effectiveness & problems: From this example it is clear that ID3 is rather fast and will be able to classify any of the training examples correctly. Quinlan noted that the expected time complexity of the algorithm lies in $\mathcal{O}(|A| * |T|)$ where $|A|$ describes the amount of attributes in T [4, pg.92].

However, ID3 also faces problems, that cannot be ignored:

- Only deals with discrete data
  - Continuous data has to be quantized before working on it. This can lead to problems in the classification since the data has to be effectively changed
- Cannot handle missing values
  - The training examples cannot contain any missing values which means, that the examples have to be prepared for the algorithm making data processing more complicated
- Susceptible to high noise
  - If the data contains high noise levels (which is a wrong labelling of attributes) ID3 will have an error rate of up to 50% [4, pg.95 (table)].

D. C4.5

In order to combat these problems C4.5 was developed as an extension to ID3. At its core C4.5 works just as ID3 as explained by Quinlan himself [5]. Major improvements have

been summarized by Rokach and Maimon [2, pg.78], which will be explained in the following sections.

1) Continuous data: C4.5 can handle continuous data by trying to discretize the continuous attribute into 2 classes based on a threshold. Suppose the attribute a contains the values $a_0, a_1, ..., a_n$, the algorithm will try to find an index i so that $t = \frac{a_i + a_{i+1}}{2}$ will yield the best information gain when splitting the attribute into $a \leq t$ and $a > t$ [5, pg.78].

In newer versions of C4.5, this works differently [5]:

- The information gain for an attribute is reduced by $log_2(n-1)/|T|$ (where $n-1$ is the number of possible thresholds)
- The threshold is chosen to maximize gain while the attribute is chosen by adjusted gain ratio

2) Missing values: The algorithm now handles missing values by giving them the name ? in order to work with them.

3) Pruning: C4.5 has additional pruning after tree creation in order to cut down on tree size and remove nodes that are not relevant for the actual classification process. Pruning will be discussed in chapter IV-A.

E. Other algorithms

There are other induction algorithms that will be discussed in a short manner.

1) CART: The CART (meaning *Classification and Regression Trees*) algorithm has the distinct advantage of being able to not only produce classification-, but also regression trees. It uses a splitting criterion based on the Gini-index, which "measures the divergences between the probability distributions of the target attributes values" [2, pg.62].

2) CHAID: The *Chi squared Automatic Interaction Detection* algorithm uses statistical tests in order to find attributes to do the splitting on. It also uses additional stopping criteria such as a *maximum tree depth*.

3) QUEST: The *Quick, Unbiased, Efficient Statistical Tree* also uses statistical tests (chi-squared tests and variance analysis [3, pg.15]) as a splitting criterion and tries to be as unbiased as possible. It will always yield a binary decision tree and use *clustering techniques* in order to generate two classes in a multinominal dataset [2, pg.80].

III. Evaluation techniques

This chapter will focus on evaluating decision trees, giving measurements for errors and discuss other problems that decision trees face in a supervised learning task such as classification.

A. Measurements

1) Generalization Error: The generalization error is a measurement in order to understand whether the tree can generalize a concept about the underlying data. Defining this generalization error is hard, since there is no mathematical definition of a *general rule* for the data. In order to evaluate this, the distribution of the underlying dataset has to be known, which is not often the case [2, pg.31].

2) Misclassification Rate: This measure can easily be calculated by simply testing the classifier with a testing dataset. Let $|X|$ denote the amount of testing examples, DT denote a decision Tree and let $|DT_W|$ be the amount of wrongly classified examples of X. The measure is defined as:

$$R(DT, X) = \frac{|DT_W|}{|X|}$$

3) Other measurements: In binary classification tasks (such as making a *True or False* classification) measurements can be defined on true and false positives and negatives [2, pg.34]. In the following notation positive and negative samples are denoted with p and n. The t and f prefixes indicate a $true$ or $false$.

$$Sensitivity = \frac{tp}{p}$$

$$Specificity = \frac{tn}{n}$$

$$Precision = \frac{tp}{tp + fp}$$

$$Accuracy = Sensitivity * \frac{p}{p+n} + Specificity * \frac{n}{p+n}$$

Precision takes false positives in effect to have a way of measuring if the classifier tends too heavily in a positive or negative side. *Accuracy* takes in effect, how well the classifier does with classifying positives and negatives respectively and weighs these abilities with the amount of positives and negatives in the data.

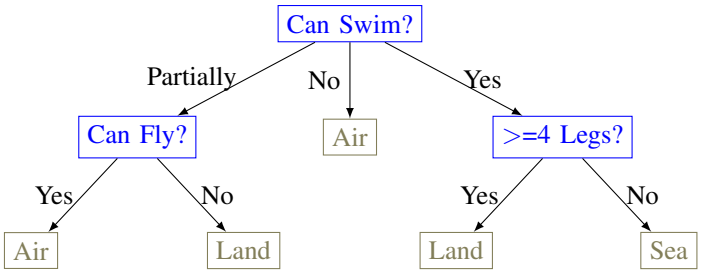


Fig. 3: Decision tree from noisy data

B. Robustness and Stability

In order to be robust, the algorithm has to be able to handle noise in data. Thus randomly changed attributes should not have a huge impact on the tree itself if the changes are not to frequent. A demonstration of noisy data is table III, which is the same table as table I but with small modifications. The resulting tree using ID3 is shown in figure 3. Looking at this tree it is easy to see that not only did it create more nodes in a different order but also still has a valid classification for the examples from table I. In this case the small noise did not create an error. Therefore the robustness of ID3 was superb in this instance. More noise on the other hand (especially noise in the target class attribute) can have worse effects on the trees performance [4, pg.95 (table)].

Stability is the property of the algorithm to produce the same tree over and over again with different data from the same training set. Imagine the training examples from table I are split up into all examples with *Sea* and *Land* classes and another set with all *Air* and *Land* animals. One tree will have no *Air* and the other one will have no *Sea* class. When adding the data point from table II to the table I, the resulting tree is different from the tree before since the generalized rule that all animals with less than 4 legs belong to the *Sea* class is obviously wrong.

TABLE II: Extra data point

Name	Can fly?	Can swim?	>=4 Legs?	Targeted class
Turtle	No	Yes	Yes	Sea

TABLE III: Noisy training examples

Name	Can fly?	Can swim?	>=4 Legs?	Targeted class
Sparrow	***No***	No	No	Air
Parrot	Yes	No	No	Air
Kingfisher	Yes	Partially	No	Air
Eagle	Yes	No	No	Air
Shark	No	Yes	No	Sea
Dolphin	No	Yes	No	Sea
Whale	***Yes***	Yes	No	Sea
Bear	No	Yes	Yes	Land
Dog	No	Yes	Yes	Land
Cat	No	Partially	***No***	Land

C. Overfitting

Overfitting is the simple concept of a classifier losing its ability to learn general rules about the data and focus to much on the training examples. Sadly, this is something happening often with decision trees, since they have very strict and discrete rules. An example of overfitting can be observed in our tree in figure 2c. Imagine the data point from table II is used to test the data. It will obviously be classified as a *Land* animal even though it should be a *Sea* animal. This is due to the fact that the original training examples had no instance of a *Sea* animal having more than four legs. Another example of overfitting is the tree in figure 3. Suppose we want to classify a giraffe which cannot swim. The tree would classify it as a flying animal.

Overfitting is related to the number of nodes in the tree. Not enough nodes and also too many nodes will lead to overfitting [2, pg.57]. Also related to this issue is *underfitting* which is the exact opposite. It describes not focusing on the rules in the data enough. It can occur when pruning a decision tree too much since it is possible to cut out a lot of *classifying* information.

IV. Further topics

In this chapter we will look at advanced topics in decision trees, which is *pruning* and *ensemble learning* in form of *random forests*.

A. Pruning

Instead of growing the tree and cutting it down while growing, one could grow a rather large tree and cut it down afterwards. Since the tree can be tested while cutting it down it could lead to favourable results.

The pruning method discussed here is called *minimal cost-complexity* and was described by Breiman, Friedman, Stone and Olshen [6, pg.61]. It defines the following measurement, called the *cost-complexity measurement*, for a dataset X and a tree T where $|T|$ defines the amount of nodes:

$$R_\alpha(T, X) = R(T, X) + \alpha|T|$$

In this case α is ≥ 0 and is called a complexity parameter. $R(T)$ is the *misclassification rate* of the tree defined in chapter III-A.2.

When pruning the tree the algorithm tries to minimize this measure. Its obvious that a large number of nodes in a subtree is punished for a high value of α. Therefore the algorithm will search for subtrees with a high misclassification error or a high number of node. Of course it is complicated looking at the number of nodes since it is heavily dependent on the number of attributes in T. Hence there has to be the parameter α in order to find the right *punishment* for large tree size.

B. Ensemble Learning

When trying to improve stability in decision trees one can try to use the concepts of *bagging and boosting*. *Bagging* is the concept of building different classifiers C^i and aggregating them into one classifier C^*. In the context of decision trees, the trainings set T would be split into different sets T^i that do not have to be disjunctive. Based on these sets different decision trees DT^i are build. When classifying information every decision tree will make its own classification c^i and these will be summarized by returning the class that is returned by most of the trees. A visual representation with 4 individual classifiers is shown in figure 4.

Boosting is similar to bagging but uses weights in a more complicated training process. Quinlan demonstrated that C4.5 works well with bagging and boosting [7].

Based on this work Breiman [8] defined and studied random forests. A very simple approach of creating such a random forest is to create multiple decision trees and do the attribute split completely randomly. This ensures that the system has no bias whatsoever and should ideally make the classifier more stable and robust. Breiman described more ways of creating such a random forest and argued that *Adaboost* a boosting algorithm can also be considered a random forest.

V. Broader view on data mining

This chapter will try to compare decision trees with other data mining techniques and showcase its strengths as well as weaknesses. It is important to note that there is not *one* decision tree algorithm but there are innumerous algorithms that create such a tree. Therefore the following discussion is about decision trees in general.

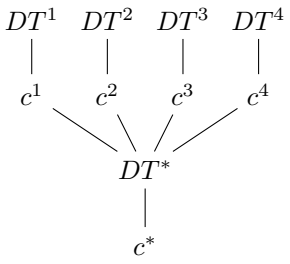


Fig. 4: Bagging example with 4 classifiers

A. Differences compared to other techniques

1) Advantages: The big advantage of decision trees is the easy readability since one can always make a visual representation of the build decision trees such as the example in figure 1. Rokach and Maimon noted that this comprehensibility is completely ignored in other classifiers such as neural networks or support vector machines [2, pg.52-53]. Decision trees, according to Kotsiantis [9, pg.263 (table)], are the best technique when it comes to handling different kinds of data (continuous , discrete or binary), even better than neural networks and also have a strong tolerance when it comes to missing values or irrelevant attributes. The latter can be shown in figure 2c where the irrelevant attribute *Can Swim?* was ignored in the tree.

2) Disadvantages: The disadvantages have already been demonstrated in chapter III where overfitting and missing stability were demonstrated. In general, decision trees are not very tolerant to noise. They are also not very good for incremental learning tasks like recurrent neural networks are.

B. An Outlook

Even with neural networks on the rise, decision trees are still used in recent research. Especially random forests are used heavily in order to combine other models into an ensemble learning scheme. Lin, Wang, Xie and Zhong [10] demonstrated a mixture of neural networks in combination with a random forest, built by the CART algorithm, in order to do time series prediction. This is one of many examples where decision trees are effectively used. A state of the art machine learning library for python, called *scikit-learn* [11], also features decision trees and random forests.

VI. Conclusion

It has been demonstrated that decision trees are a powerful tool for data mining. Tasks like classification and regression can not only be realized but also visualized by decision trees. Therefore it is easy to understand their inner working and to evaluate them by *taking a good luck*. In general, their function is very intuitive. Even though they do not have the performance of far more complex methods, they are easier to compute and therefore worth studying.

References

[1] U. Fayyad, G. Piatetsky-shapiro, and P. Smyth, "From data mining to knowledge discovery in databases," *AI Magazine*, vol. 17, pp. 37–54, 1996.

[2] L. Rokach and O. Maimon, *Data Mining with Decision Trees: Theroy and Applications*. River Edge, NJ, USA: World Scientific Publishing Co., Inc., 2008.

[3] W. Loh, "Classification and regression trees," *WILEY INTERDISCIPLINARY REVIEWS-DATA MINING AND KNOWLEDGE DISCOVERY*, vol. 1, no. 1, pp. 14–23, JAN-FEB 2011.

[4] J. Quinlan, "Induction of decision trees," *MACH. LEARN*, vol. 1, pp. 81–106, 1986.

[5] J. Quinlan, "Improved use of continuous attributes in c4.5," *Journal of Artificial Intelligence Research*, vol. 4, pp. 77–90, 1996.

[6] L. Breiman, J. Friedman, C. Stone, and R. Olshen, *Classification and Regression Trees*, ser. The Wadsworth and Brooks-Cole statistics-probability series. Taylor & Francis, 1984. [Online]. Available: https://books.google.de/books?id=JwQx-WOmSyQC

[7] J. Quinlan, "Bagging, boosting, and c4.5," in *In Proceedings of the Thirteenth National Conference on Artificial Intelligence*. AAAI Press, 1996, pp. 725–730.

[8] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[9] S. Kotsiantis, "Supervised machine learning: A review of classification techniques," *Informatica (Ljubljana)*, vol. 31, no. 3, pp. 249–268, 2007.

[10] L. Lin, F. Wang, X. Xie, and S. Zhong, "Random forests-based extreme learning machine ensemble for multi-regime time series prediction," *Expert Systems with Applications*, vol. 83, pp. 164–176, 2017.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.