|  |
| --- |
| UNS |
| ARFITEC – Data project VygLab |
| Geology |

|  |
| --- |
| Robin Masliah  29/11/2106 |

Summary

[State of art 1](#_Toc468285268)

[Goal 2](#_Toc468285269)

[Hypothesis 2](#_Toc468285270)

[Definitions and recall 2](#_Toc468285271)

[Build the study base 3](#_Toc468285272)

[Technology used and execution 4](#_Toc468285273)

[Process 5](#_Toc468285274)

[Choose of algorithm 6](#_Toc468285275)

[Output 6](#_Toc468285276)

[Conclusion 8](#_Toc468285277)

# State of art

The Geology Department of the UNS owns a certain quantity of data of different families of minerals. These data are presented in 6 different excel files. Each of the 6 files contains different samples of a single mineral family. The 6 distinct families are: Alaskan, Alkali-lamp, Komatites, Layered, Ophiolites and Xenoliths.

A row is a sample and the columns are the properties associated to the sample. These are the percentages of the chemical elements of the samples (eg: SiO2, TiO2, Al2O3, etc.) and also the end-members (eg: MgCr2O4, FeCr2O4, MnCr2O4, etc.).

The chemical elements represent the chemical composition of each sample and figure also in the files their end-members. An **endmember** (also **end-member** or **end member**) in mineralogy is a mineral that is at the extreme end of a mineral series in terms of purity. Minerals often can be described as solid solutions with varying compositions of some chemical elements, rather than as substances with an exact chemical formula. There may be two or more endmembers in a group or series of minerals.

Normally, geologists analyze these minerals considering combinations of end-members by using a prism representation. The data whom will appear in the prism must be representative of the family of the sample. We want to analyze all the chemical elements and end-members to confirm that these are really the most relevant.

# Goal

We want to analyze all the chemical elements and end-members to verify that these are really the most relevant. The representation will be a prism. So that we must choose 6 + 2 columns susceptible to be the elements the most representative of a sample.

# Hypothesis

To determine which chemical element data is relevant we will try to determinate what chemical element of the chemical composition and what end-members define a sample and allows to characterize a mineral family. So that, we will build a program who tries to determine the family of a sample according to its chemical composition.

Data science is all designated to solve this problem. It is well used to make scoring analysis based on statistical research and machine learning algorithms.

# Definitions and recall

A score allows to give to someone or something a grade estimating the probability that an event happens.

Nowadays the scoring method is well used in marketing studies because it helps to have an optimized target of a commercial solicitation. Never the less, we will see how we can apply this solution to our problems with minerals.

The scoring step by step

First, to make a score, we have to build the base that we want to study.

1. Definition of the event we want to study;
2. Definition of the period of the study;
3. Construction of the variables;
4. Separation of the data (make a train set and a validation set).

Second, we have to make the modelisation.

1. Build several models;
2. Choose the « best » model;
3. Make an interpretation of the model chosen.

Third, we need to exploit the welfare of the scored generated.

1. Implement it in a management tool or a database to put it in production;
2. Follow and find more accuracy of the model.

Fourth, we can realize different graphical visualization.

1. Choose a relevant form of visualization;
2. Coordinate or manipulate the data to make reading understandable;
3. Put the data into a graphical representation.

# Build the study base

With all data from the 6 files let’s build one data file. We treat here only CSV files.

The idea here is to train a machine learning algorithm to take decisions or to make choices. The choice represents in our case the ability of recognizing the family of a sample according to its mineral composition and/or its end-members.

I transformed the dataset in a supervised dataset by giving every family a number in the following order:

1 Alaskan

2 Alkali-lamp

3 Komatites

4 Layered

5 Ophiolites

6 Xenoliths

This variable is called id\_stone.

Now the problem is supervised. To treat this data, we can use trees machine learning algorithms. This fits in general and it’s supposed that an appropriate one to our dataset could be founded.

Once the data is ready, we need to construct a train set and a cross validation set of data. To construct the train set there is two ways. One of them could be to use a python function who chooses randomly the tuples. It is even possible to choose the quantity of data we want to put in the train set and the validation set. This was the first idea but the limit is that it makes the interpretation uncomfortable because it mixes all the lines and we lose the advantage of the order of the families of the minerals.

So the method I finally decided to adopt was to take around 40 tuples of every family manually and separate the dataset into two new CSV files.

It is also important to manage the problem of missing values. Some lines have missing values and could generate errors during execution. So that, we can put the value « NaN » or attribute a 0 if it doesn’t make the prediction the prediction. In our case it doesn’t.

With this alternative we have the values below:

|  |  |  |  |
| --- | --- | --- | --- |
| Family name | Composition + EM | Id\_stone (number corresponding to a family) | Prediction obtained |
| Kromatites | MgCr2O4, FeCr2O4, MnCr2O4 | 1 | 1 |
| Kromatites | MgCr2O4, FeCr2O4, MnCr2O4 | 1 | 1 |
| Kromatites | MgCr2O4, FeCr2O4, MnCr2O4 | 2 | 2 |
| Kromatites | MgCr2O4, FeCr2O4, MnCr2O4 | 2 | 3 |
| Kromatites | MgCr2O4, FeCr2O4, MnCr2O4 | 2 | 2 |

What we would have by choosing randomly the tuples for the train set:

|  |  |  |  |
| --- | --- | --- | --- |
| Family name | Composition + EM | Id\_stone (number corresponding to a family) | Prediction obtained |
| Kromatites | MgCr2O4, FeCr2O4, MnCr2O4 | 6 | 6 |
| Kromatites | MgCr2O4, FeCr2O4, MnCr2O4 | 1 | 1 |
| Kromatites | MgCr2O4, FeCr2O4, MnCr2O4 | 4 | 2 |
| Kromatites | MgCr2O4, FeCr2O4, MnCr2O4 | 5 | 3 |
| Kromatites | MgCr2O4, FeCr2O4, MnCr2O4 | 2 | 2 |

# Technology used and execution

Python 4.4

Framework Anaconda 3

Environment Notebook Jupyter

To install all the environment first download and install Anaconda.

<https://www.continuum.io/downloads>

Then launch the Jupyter Notebook installed thanks to Anaconda. Then create a file to code .ipynb. The file will belong to the tree of files and it has to be saved.

# Process

In order to proceed with the classification we proceed as it is detailed. First, we split the dataset data into two others datasets called train and test. Train represents 80% of all the amount of data we have and test represents the rest.

With the variable id\_stone contained in train we (siempre usar primera persona del singular o del plural; no mezclar) create the data frame « target ». This is the goal we want to approach with the test dataset.

We calculate the importance of each features in the different machine learnings algorithms. Then we build the score based on the features. We will further explain the different algorithms used.

Let’s do the same calculous on the main composition data et all data together. We note that the recognition is 3 percent more precise with all the data processed together.

* Construction of the target by stocking the values:

*target = train["id\_stone"].values*

* Insertion of the features to analyze:

*features\_one =* ***train****[["SiO2", "TiO2", "Al2O3", "Cr2O3", "V2O5", "Fe2O3", "FeO", "MnO", "MgO", "CaO", "ZnO", "NiO", "Na2O", "K2O", "MgAl2O4", "FeAl2O4", "MnAl2O4", "ZnAl2O4", "MgFe2O4", "FeFe2O4", "MnFe2O4", "ZnFe2O4", "NiFe2O4", "MgCr2O4", "FeCr2O4", "MnCr2O4", "ZnCr2O4", "NiCr2O4", "MgV2O4", "FeV2O4", "MnV2O4", "Mg2TiO4", "Fe2TiO4"]].****values***

* Insertion of the test features to process with the machine learning algorithm :

*features\_forest =* ***train****[["SiO2", "TiO2", "Al2O3", "Cr2O3", "V2O5", "Fe2O3", "FeO", "MnO", "MgO", "CaO", "ZnO", "NiO", "Na2O", "K2O", "MgAl2O4", "FeAl2O4", "MnAl2O4", "ZnAl2O4", "MgFe2O4", "FeFe2O4", "MnFe2O4", "ZnFe2O4", "NiFe2O4", "MgCr2O4", "FeCr2O4", "MnCr2O4", "ZnCr2O4", "NiCr2O4", "MgV2O4", "FeV2O4", "MnV2O4", "Mg2TiO4", "Fe2TiO4"]].****values***

* Call of the algorithm:

***forest*** *=* ***RandomForestClassifier****(max\_depth = 10, min\_samples\_split=2, n\_estimators = 100, random\_state = 1)*

* Processing of data comparing to the target:

*my\_forest = forest.fit(features\_forest,* ***target****)*

* Collect of the features importance:

*print(my\_forest.feature\_importances\_)*

* Build and display the score applied to be validation set called « **test** »:

*test\_features =* ***test****[["SiO2", "TiO2", "Al2O3", "Cr2O3", "V2O5", "Fe2O3", "FeO", "MnO", "MgO", "CaO", "ZnO", "NiO", "Na2O", "K2O", "MgAl2O4", "FeAl2O4", "MnAl2O4", "ZnAl2O4", "MgFe2O4", "FeFe2O4", "MnFe2O4", "ZnFe2O4", "NiFe2O4", "MgCr2O4", "FeCr2O4", "MnCr2O4", "ZnCr2O4", "NiCr2O4", "MgV2O4", "FeV2O4", "MnV2O4", "Mg2TiO4", "Fe2TiO4"]].****values***

*my\_prediction = my\_forest.predict(test\_features)*

*print(my\_prediction)*

# Choice of algorithm

With a supervised dataset, the efficient algorithms to make predictions are the trees. We will try 2 different varieties of tree algorithms:

**Decision tree**

This method generates production rules under the form of a tree with branches. This makes the method easy for exploitation.

The goal is to divide the populations successively in sub-groups according to the different values of the variables. Every step the tree makes a discrimination of the continuous values and build a rule.

**Decision tree** by changing deep research parameters.

*#Control overfitting by setting "max\_depth" to 10 and "min\_samples\_split" to 5 : my\_tree\_two*

*max\_depth = 10*

*min\_samples\_split = 5*

*my\_tree\_two = tree.DecisionTreeClassifier(max\_depth = 10, min\_samples\_split = 5, random\_state = 1)*

*my\_tree\_two = my\_tree\_two.fit(features\_two, target)*

*#Print the score of the new decison tree*

*print(my\_tree\_two.score(features\_two, target))*

*print(my\_tree\_two.feature\_importances\_)*

Here the max depth of research of the algorithm was changed. It means that the research into the will ¿Qué significa? be restricted after reaching a certain depth.

**Random Forest** which is a succession of tree algorithm. (Explicar un poco más)

# Output

In our case we will consider precisely the features´ importance. The obtained percentages give us an indication of the weight of a feature to help the algorithm to make the decision. In an Excel file are all the features importance and colored the 8 biggest.

Features importance: Acomodar table al ancho de la hoja Además, explcar qué es DT1, DT2 y poner en claro qué es RF. En algún lado debería decir que Random Forest es RF y que Decision Tree es DT (podés ponerlo arriba cuando los nombrás). Pero además debe explicarse por qué ponés DT1 y DT2

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Finals members |  |  |  |  |  |  |  |
|  | MgAl2O4 | FeAl2O4 | MnAl2O4 | ZnAl2O4 | MgFe2O4 | FeFe2O4 | MnFe2O4 |
| RF | 8.58% | 7.58% | 6.72% | 2.57% | 6.52% | 8.34% | 6.76% |
| DT1 | 10.30% | 8.34% | 5.64% | 0.00% | 9.08% | 0.00% | 8.29% |
| DT2 | 9.13% | 6.82% | 5.79% | 0.00% | 5.91% | 0.95% | 6.90% |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Composition |  |  |  |  |  |  |  |
|  | SiO2 | TiO2 | Al2O3 | Cr2O3 | V2O5 | Fe2O3 | FeO |
| RF | 15.74% | 14.26% | 11.93% | 10.50% | 3.16% | 8.13% | 12.10% |
| DT1 | 27.80% | 25.18% | 21.05% | 1.07% | 1.05% | 3.02% | 6.12% |
| DT2 | 28.57% | 24.10% | 24.51% | 3.65% | 0.00% | 2.75% | 1.91% |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EM+Compositon | SiO2 | TiO2 | Al2O3 | Cr2O3 | V2O5 | Fe2O3 | FeO | MnO | MgO |
| RF | 10.73% | 7.02% | 5.77% | 4.70% | 0.80% | 2.54% | 4.39% | 3.50% | 6.48% |
| DT1 | 17.36% | 4.34% | 11.63% | 1.86% | 0.00% | 4.09% | 0.00% | 0.52% | 0.00% |
| DT2 | 18.09% | 1.04% | 14.70% | 0.00% | 0.00% | 1.12% | 0.00% | 3.02% | 0.00% |

Then we calculate the AUC which represent the probability to put a positive in front of a negative (?). It indicates the accuracy of the score.

For RF : AUC = 0.416666666667

For DT1 : AUC = 0.544444444444

For DT2 : AUC = 0.57962962963

According to the results, we can assume that the Decision tree 2 was the better algorithm. We can see that the performance of the second decision tree is higher.

Random forest is a well-known classifier. In its simplest form it can be thought of using bagging and Random Subsets meta classifier on a tree classifier. We wish to improve the performance of a tree classifier by averaging or voting between several weak tree classifiers. However, we can't just build multiple trees on the same data as we will just get the same results. We introduce randomness of two types: each tree is built on slightly different rows sampled with repetitions from the original (bagging) and columns each tree (or in some cases each branch decision) is built using a small randomly selected subset of columns. Esta explicación deberáestar más arriba

Here we are only considering algorithms classifier.

Nevertheless, in our case, it is not necessary to use a random election. A random forest is just a collection or ensemble of decision trees.

A decision tree is built using the whole dataset considering all features, but in random forests a fraction of the number of rows is selected at random and a particular number of features are selected at random to train on and a decision tree is built on this subset.

Similarly a number of decision trees are grown, each will probably have a different subset since it is being randomly selected and hence each decision tree will be different, and each tree will vote for a particular class and the class which gets maximum number of votes is the predicted class.

By changing the parameters of research it should be enough to enhance the accuracy of the calculous.

# Conclusion

To conclude we see that by training the second algorithm of decision tree we arrive to have 58% of recognition. This number tends to be enhance if we change the parameters of research. Also, the datasets can be changed and it would also affect the precision of the calculous. We can notice the algorithm is effectively learning by watching the different AUC.

The AUC seems to show a difference on the same datasets for every algorithm so we have to confirm that when machine learning is applied to the cross validations set it has already been trained.

Besides, for every algorithm, we quote only a few difference of the features importance. This confirms that the algorithms Random Forest and Decision tree belong to the same « family ».

Corregir inforne

Los archivos tienen aue tener el mismo nonbre que en el informe

Mandarnos el informe y los archivos correspondientes, incluido el codigo