ExampleBasedExplanations :

This class helps the user to understand and explain the behaviour of a machine learning model, by returning instances of the dataset that affect the prediction of the ML.

You can import the library into your code as follow:

```
import ExampleBasedExplanations
```

After importing the above mentioned class, please create an instance of it and use that instance to call the below mentioned functions.

For Example: `test=ExampleBasedExplanations.ExampleBasedExplanations()`

Note: In the packaged folder, we have kept an **Example.py** file which has examples of calling the different functions and also datasets we have worked on.

Methods:

For tabular data type there are 2 methods you can use:

1. **KNNTabularOutlie**r:
   This method returns the data after removing the outliers.
   - Arguments:
     - file(Datatype:String): path to the file of the dataset
   - Returns: dataset without outliers

2. **KNNTabularNeighbors**:
   This method returns the k neighbors of a sample from a tabular dataset.
   - Arguments:
     - dataset(Datatype:String): path to the file of the dataset. Or the returned dataset from the KNNTabularOutlier().
     - k(Datatype:Integer): , number of neighbours you want to get back.
     - sample: list, an instance in the same structure of the dataset features.
   - Returns : list of features from the dataset

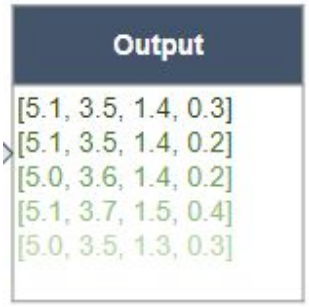
Example to call the methods for tabular data :

File_name = ‘iris.csv’

`test.`KNNTabularOutlier(file_name)

`test.`KNNTabularNeighbors(file_name, k, 5, [5.4,3.9,1.7,0.4])

Output:



Libraries Required for KNN RDF:

- Numpy
- Pandas
- rdflib
- Sklearn
- networkx
- Counter(from "**collections**" package which usually is pre installed)
- defaultdict(from "**collections**" package which usually is pre installed)
- Operator
- itertools

For RDF Dataset there are 2 methods you can use:

1. **knnRDF**:
   This method creates a graph representation of rdf dataset and calculates nearest neighbors to the given input node by using Jaccard coefficient as metrics.
   - Arguments:
     - file(Datatype:String): It specifies the path where the rdf file is located. Example:"F:/semanticLab/dataset/WikiMovie.rdf"
     - number_of_neighbors(Datatype:int): required number of nearest neighbors.
     - node(Datatype:String): It is the input node for which the nearest neighbors are calculated. ForExample:'http://www.semanticweb.org/vinu/ontologies/2014/6/untitled-ontology-91#naomi_watts'.
   - Returns: Dictionary of nodes and their similarity
2. **knnRDFCluste**r:
   this function is used to create clusters
   - Arguments:

        - file(Datatype:String): It specifies the path where the rdf file is located. Example:"F:/semanticLab/dataset/WikiMovie.rdf"
    - Returns: Set of nodes present in the cluster.

Example to call the methods for Knn RDF:

```
File_name = "F:/semanticLab/dataset/WikiMovie.rdf"
node='http://www.semanticweb.org/vinu/ontologies/2014/6/untitled-ontology-91#naomi_watts'
number_of_neighbors=3
test.knnRDF(File_name,node, number_of_neighbors)
test. knnRDFCluster(file_name)
```

Note: KNN on the rdf dataset works only for fully connected rdf graphs,Please choose the dataset accordingly.

Libraries Required for ProtoDash:

- numpy
- rdflib
- pyspark
- qpsolvers
- mnist

We implemented a prototype selection algorithm proposed by Gurumoorthy, K. S. et al.[1] called ProtoDash. We used functions from an open-source library AI Explainability 360[2] in our code

For RDF data type using ProtoDash algorithm:

- **protoDashRDF**:
  The method returns prototypes (neighbours) that can be thought as the cluster which the sample belongs to. Or vice versa, the sampled datapoint can be thought as cluster centroid, and the explaining prototypes as the data that belong to that cluster.
    - Arguments:
        - file_name (Datatype:String): It specifies the path where the rdf file is located. Example:"F:/semanticLab/dataset/WikiMovie.rdf"
        - num_proto (Datatype:int): required number of prototypes.
        - sample_triple (Datatype:String): the triple to be given to the ProtoDash algorithm as a dataset to be explained.

[1] "Efficient Data Representation by Selecting Prototypes with ...." 12 Aug. 2019, https://arxiv.org/abs/1707.01212. Accessed 1 Mar. 2021.
[2] "Trusted-AI/AIX360: Interpretability and explainability of data ... - GitHub." https://github.com/Trusted-AI/AIX360. Accessed 1 Mar. 2021.

Example:"http://www.semanticweb.org/vinu/ontologies/2014/6/untitled-ontology-91#jimmy_sangster,http://www.w3.org/1999/02/22-rdf-syntax-ns#type,http://www.semanticweb.org/vinu/ontologies/2014/6/untitled-ontology-91#Film_writer"

- Returns: a list of pairs of weight and the prototypes for the sample_triple on a plot (Y-axis: prototypes; X-axis: importance weights, which are indicators of the similarity).

Example to call the method:

num_proto = "4",
sample_triple = "http://www.semanticweb.org/vinu/ontologies/2014/6/untitled-ontology-91#jimmy_sangster,http://www.w3.org/1999/02/22-rdf-syntax-ns#type,http://www.semanticweb.org/vinu/ontologies/2014/6/untitled-ontology-91#Film_writer"
protoDashRDF(file_name , num_proto , sample_triple)

For image data type using ProtoDash algorithm:

- **ProtoDashOnImage**:
  This method selects prototypes from a source dataset (created from an imported MNIST image dataset library) that are representative of a given target dataset.
  - Attributes:
    - digit (Datatype:int): a digit between 0-9 which we want to find prototypes for.
    - num_proto (Datatype:int): number of prototypes for ProtoDash to select from the source dataset
  - Returns: the images of prototypes for the digit.

Example to call the method:

digit = “0”,
num_proto = “3”
protoDashImage(digit, num_proto)

How to configure environment for ProtoDash:

System requirements

16GB RAM is required. With 8GB RAM we can get results from a small dataset but the result is not satisfiable.

Also, please install C++ Build Tools from here. In order to install the **qpsolvers** package (which is used in the code) in python we need to install this beforehand.

Installing Prerequisites (for Windows)

We implemented the algorithm using Apache Spark via the API PySpark, so we need to install prerequisites in order to run the code. We are assuming that you have on your PC a Python version at least 2.6.

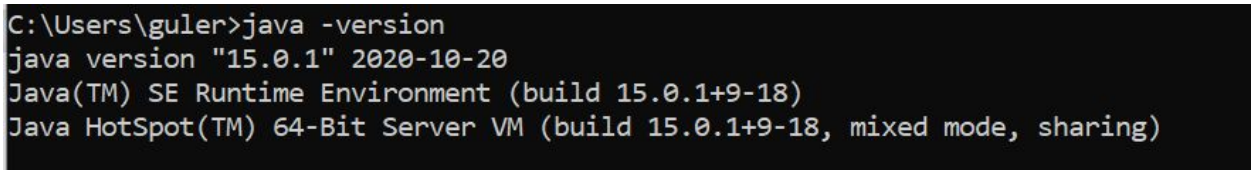
PySpark requires Java version 7 or later.

1. Installing Java

To check if Java is available and find its version, open a Command Prompt and type the following command:

java -version

If Java is installed and configured to work from a Command Prompt, running the above command should print the information about the Java version to the console. For example, I got the following output on my laptop.



If you have an old version or it is not installed, you can download it here.

2. Installing Apache Spark

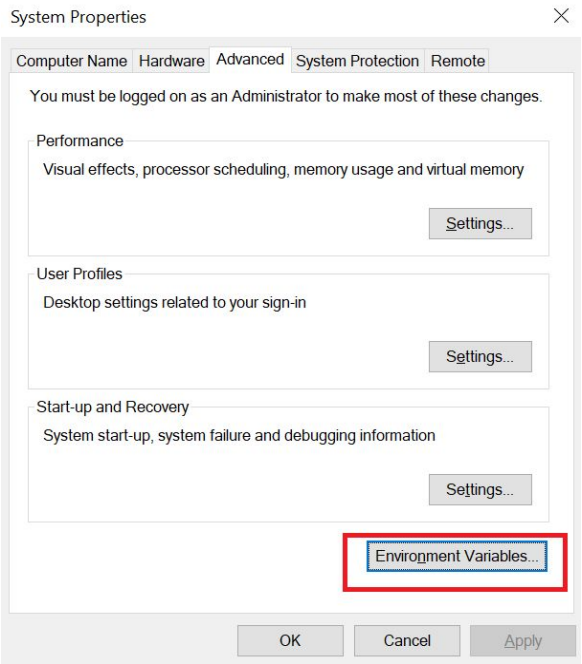
- Go to the Spark download
- Please select the recent Spark release and the Pre-built package for Apache Hadoop 2.7.
- Click the link next to Download Spark to download the **spark-3.0.1-bin-hadoop2.7.tgz**
- In order to install Apache Spark, there is no need to run any installer. You can extract the files from the downloaded zip file using winzip.
- Make sure that the folder path and the folder name containing Spark files do not contain any spaces.

- Create a folder called “spark” on your desktop and unzip the file that you downloaded as a folder called spark-3.0.1-bin-hadoop2.7. So, all Spark files will be in a folder called C:\Users\<your_user_name>\Desktop\Spark\spark-3.0.1-bin-hadoop2.7. From now on, we shall refer to this folder as **SPARK_HOME** in this document.

3. **Download winutils.exe and Set Up your Environment**

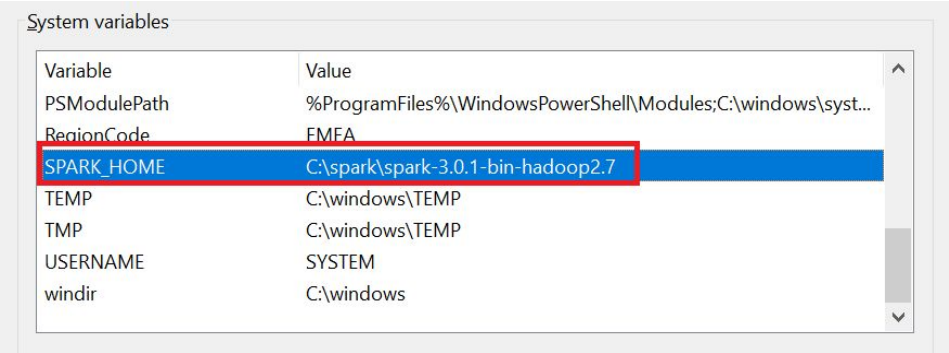
- Create a **hadoop\bin** folder inside the **SPARK_HOME** folder which we already created in the last step as above.
- Download the winutils.exe for hadoop 2.7.1 (in this case) and copy it to the **hadoop\bin** folder in the **SPARK_HOME** folder.

Now, let’s set up our environment. Go in the windows search bar and type “edit the system environment variables”.

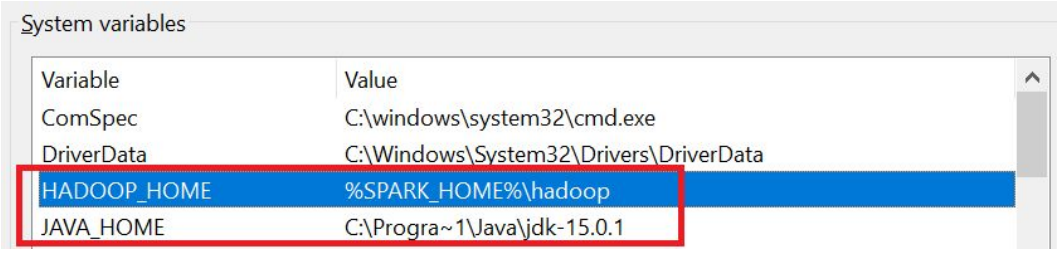


In the group of the System Variables, create three variables called **HADOOP_HOME**, **SPARK_HOME** and **JAVA_HOME** (if you already don’t have one).

- In **SPARK_HOME** → put the path of the location of the spark folder that you created before (in my case it is in **C:\spark\spark-3.0.1-bin-hadoop2.7**)

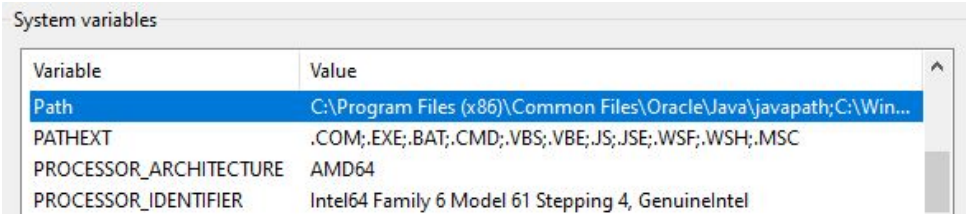


- In **HADOOP_HOME** → since the hadoop folder is inside the SPARK_HOME folder, write a value of **%SPARK_HOME%\hadoop**
- In **JAVA_HOME** → put the path of the location of your JAVA program.



Then in the variable **Path** as you can see below, we can add the following two paths:

- **%SPARK_HOME%\bin**
- **%JAVA_HOME%\bin**



To test if your installation was successful, open Anaconda Prompt, change to SPARK_HOME directory and type **bin\pyspark**. This should start the PySpark shell which can be used to interactively work with

Spark. You can exit from the PySpark shell in the same way you exit from any Python shell by typing **exit()**.

How to run the code?

ProtoDash algorithm for RDF data: To run the code import "ProtoDashRDFImage" and run the function ProtoDashRDFImage.ProtoDash.ProtoDashOnRDF().

As a 'sample_triple' you need to enter a triple which consists of subject, predicate and object (separated by comma).

Example from "WikiMovie.rdf" dataset:

```
<!-- http://www.semanticweb.org/vinu/ontologies/2014/6/untitled-ontology-91#jimmy_sangster -->
  <owl:NamedIndividual rdf:about="&untitled-ontology-91;jimmy_sangster">
    <rdf:type rdf:resource="&untitled-ontology-91;Film_screenwriter"/>
    <rdf:type rdf:resource="&untitled-ontology-91;Film_writer"/>
    <rdf:type rdf:resource="&untitled-ontology-91;Person"/>
    <untitled-ontology-91:nationality_is rdf:datatype="&xsd;string">British</untitled-ontology-91:nationality_is>
  </owl:NamedIndividual>
```

We can create the following triples:

Subject = http://www.semanticweb.org/vinu/ontologies/2014/6/untitled-ontology-91#jimmy_sangster

Predicate = http://www.w3.org/1999/02/22-rdf-syntax-ns#type

Object = http://www.semanticweb.org/vinu/ontologies/2014/6/untitled-ontology-91#Film_screenwriter

Subject = http://www.semanticweb.org/vinu/ontologies/2014/6/untitled-ontology-91#jimmy_sangster

Predicate = http://www.w3.org/1999/02/22-rdf-syntax-ns#type

Object = http://www.semanticweb.org/vinu/ontologies/2014/6/untitled-ontology-91#Film_writer

Subject = http://www.semanticweb.org/vinu/ontologies/2014/6/untitled-ontology-91#jimmy_sangster

Predicate = http://www.w3.org/1999/02/22-rdf-syntax-ns#type

Object = http://www.semanticweb.org/vinu/ontologies/2014/6/untitled-ontology-91#Person

Subject = http://www.semanticweb.org/vinu/ontologies/2014/6/untitled-ontology-91#jimmy_sangster

Predicate = http://www.semanticweb.org/vinu/ontologies/2014/6/untitled-ontology-91#nationality_is

Object = British

sample_triple = http://www.semanticweb.org/vinu/ontologies/2014/6/untitled-ontology-91#jimmy_sangster,http://www.w3.org/1999/02/22-rdf-syntax-ns#type,http://www.semanticweb.org/vinu/ontologies/2014/6/untitled-ontology-91#Film_writer