# Cognitive Systems - $2^{nd}$Assignment report

Daniel Minguez, Javier de la Rua

16th December, 2018

## 1 Introduction

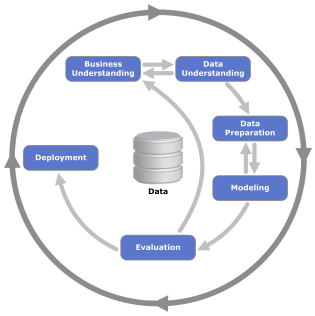
Mushrooms4all is an organization that aims to promote the collection and use of mushrooms for feeding purposes. However, this organization is aware that there are many mushrooms that are not suitable for human consumption, being dangerous to eat them since they can cause poisoning.

Although they have a quite complete database of dangerous/non-dangerous mushrooms, they aim to create a machine learning model that, based on the mushroom characteristics, allows to classify a not known mushroom species as safe or not for human consumption.

Mushrooms4all have provided their current dataset of mushrooms that are dangerous and non dangerous and in order to create a model that allows to classify a new mushroom as that can cause poisoning or not and an application to query the model.

In this assignment we are requested to put in practice the theory of data science from a model creation and deployment perspective. We are going to apply the usual methodology of a data project in order to apply and deploy a model in a given dataset:

- Investigate the data (by applying CRISP-DM methodology)
- Create models
- Evaluate the models
- Deploy models in production



## 2 Data description

We have been provided with a dataset *Mushrooms.csv*. All variables in the dataset are categorical, the dataset includes the following fields:

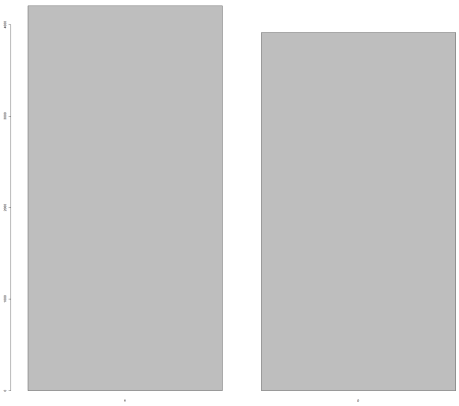
| Column | Values |
|---|---|
| veil-type | partial-p, universal-u |
| ring-number | none-n, one-o, two-t, other-b |
| bruises | no-f,bruises-t |
| habitat | grasses-g, leaves-l, meadows-m, paths-p, urban-u, waste-w, woods-d |
| stalk-surface-above-ring | fibrous-f, scaly-y, silky-k, smooth-s |
| stalk-surface-below-ring | fibrous-f, scaly-y, silky-k, smooth-s |
| cap-surface | fibrous-f, grooves-g, smooth-s, scaly-y |
| stalk-shape | enlarging-e, tapering-t |
| class | edible-e, poisonous-p |
| ring-type | cobwebby-c, evanescent-e, flaring-f, large-l, none-n, pendant-p, sheathing-s, zone-z |
| gill-spacing | close-c, crowded-w, distant-d |
| stalk-root | bulbous-b, club-c, cup-u, equal-e, missing-?, rhizomorphs-z, rooted-r |
| veil-color | brown-n, orange-o, white-w, yellow-y |
| stalk-color-above-ring | brown-n, buff-b, cinnamon-c, gray-g, orange-o, pink-p, red-e, white-w, yellow-y |
| stalk-color-below-ring | brown-n, buff-b, cinnamon-c, gray-g, orange-o, pink-p, red-e, white-w, yellow-y |
| cap-color | brown-n, buff-b, cinnamon-c, gray-g, green-r, pink-p, purple-u, red-e, white-w, yellow-y |
| gill-size | broad-b, narrow-n |
| gill-color | black-k, green-r, brown-n, buff-b, chocolate-h, gray-g, orange-o, pink-p, purple-u, red-e, white-w, yellow-y |
| spore-print-color | black-k, brown-n, buff-b, chocolate-h, green-r, orange-o, purple-u, white-w, yellow-y |
| cap-shape | bell-b, conical-c, flat-f, knobbed-k, sunken-s, convex-x |
| gill-attachment | attached-a, descending-d, free-f, notched-n |
| odor | almond-a, anise-l, creosote-c, fishy-y, foul-f, musty-m, none-n, pungent-p, spicy-s |
| population | abundant-a, clustered-c, numerous-n, scattered-s, several-v, solitary-y |

## 2.1 Exploration

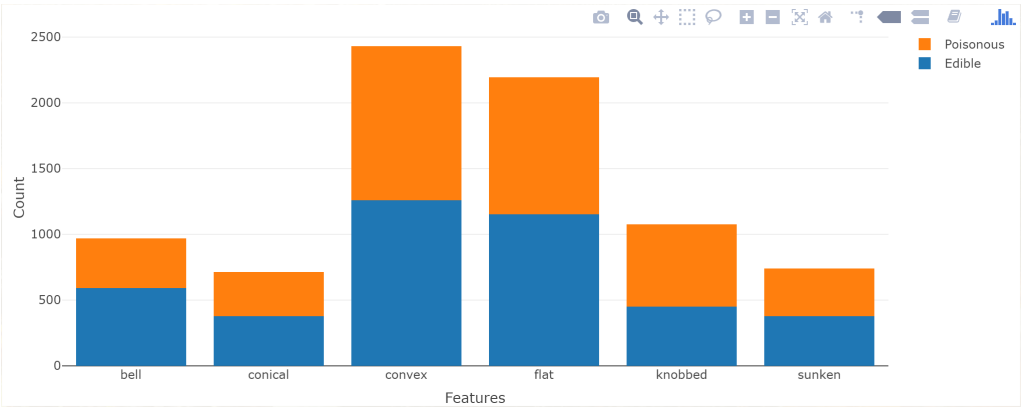
In order to perform exploration tasks, we have focus on answering different questions that we think can be interesting for this problem.

Regarding data integrity we just have found some unclassified category in ring-number that we have classified as "other".

First of all, we need to know how the class variable is distributed across the dataset in order to select subgroups if it is unbalanced. We can see that this is not the case:

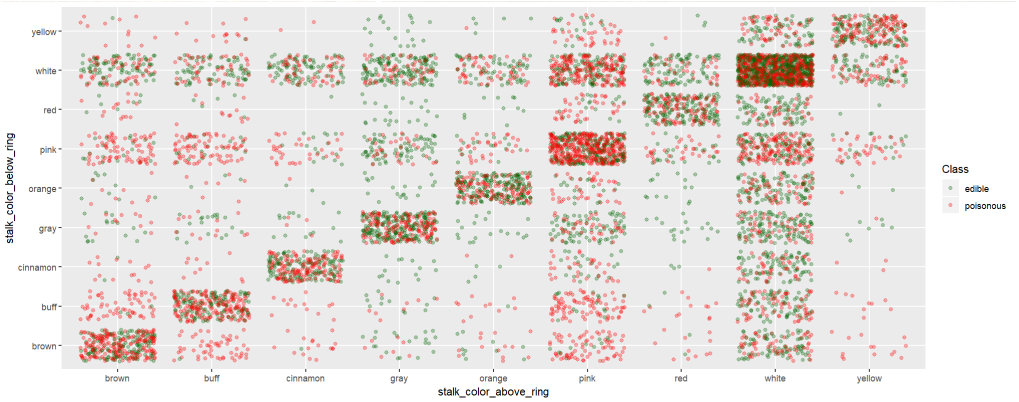


Also it would be interesting to know how the class variable is distributed across the different variables for detecting possible outliers in the observations (classes inside variables that have few observations), in order to achieve this we have included in the app the possibility to explore the different variables with a barplot (example of cap-shape):



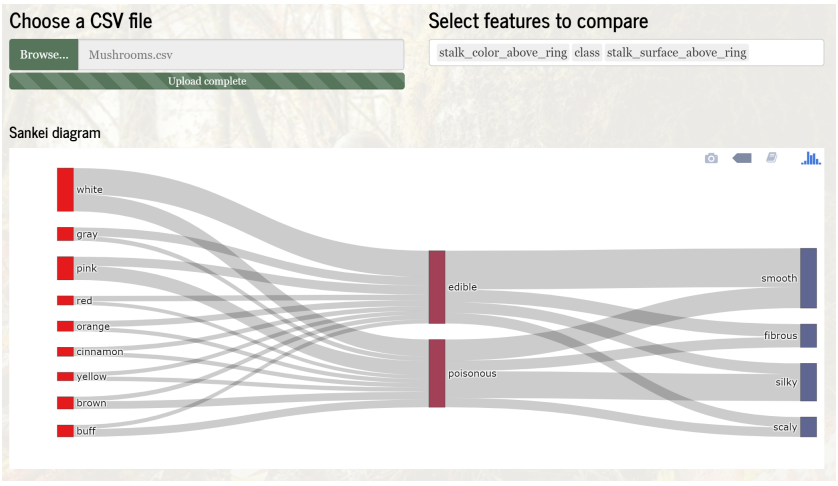
We have observed that in the class Poisonous/edible appears to be equally distributed across the different variables.

Related to this point, we also have included jitter graph in order to perform bivariate visualizations and see if the relation between variable affect the class, here is an example for the combination of colors above and below ring:




We can see that there are some combination of colors that 100% poisonous in our observations (like brown-yellow or cinamon-buff), and others that are 100% edible (like red-gray or orange-gray).

Finally we have also included a sankey diagram to explore the relation between the different variables and how they are related to each other:



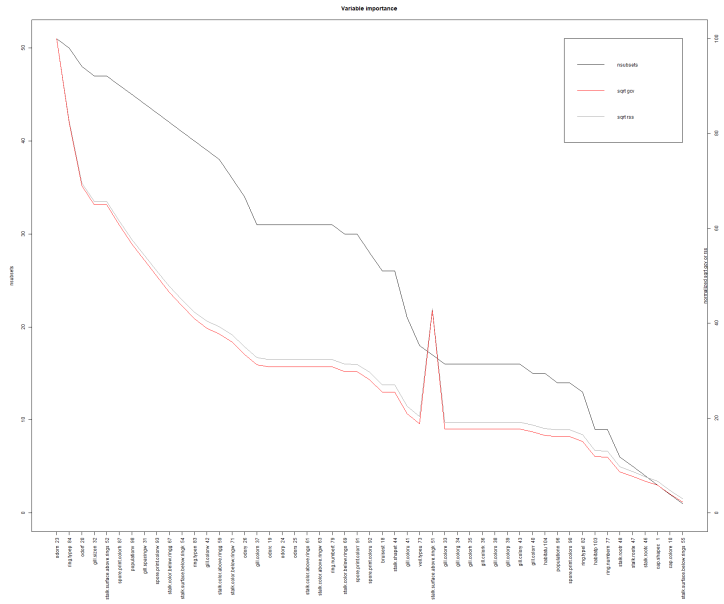
## 2.2 Preparation

In order to prepare the file for display in the application, we have replaced the codes by the description of each one.

Regarding the preparation for the model training we have split the dataset in different ways depending on the model, there is more information in the next section.

Since the models we have deployed work with categorical data, we have not applied more transformation to the dataset.

We have performed feature importance analysis in order to check if it was recommendable deleting any of the variables for applying the models. For that we have trained a gbm model from the caret library and the variable importance method from earth package that implements variable importance based on Generalized cross validation (GCV), number of subset models the variable occurs (nsubsets) and residual sum of squares (RSS). The results were the following:




After that we have tried to train some more models without variables like stal.root and gill.color, but the performance did not improve significally, so we didn´t remove any column.

As we explain in the next section, we have also tried other models that only work with numerical data, for them we have applied one hot encoding (with dummyvars function from caret library), obtaining a dataset with one column for each categorical class of each variable.

# 3 Models

As we decided to deploy some models in a Shiny server and others in AzureML studio (which theoretically allows python executed models), we were able to develop this models in different languages. We tried to cover a broad class of models of different types, using the libraries available in those languages

- **Python:**
  - Feedforward Neural network model with Keras.
  - Xgboostmodel.
  - Nearest neighbors classification using scikit learn.
- **R:**
  - H2O automl model
  - Decision tree with rpart library
  - Random forest with randomForest library
- **AzureML studio:**
  - Two-Class Boosted Decision Tree

Nevertheless due to compatibility problems with the virtual machine and time reasons we decided to implement only some of them.

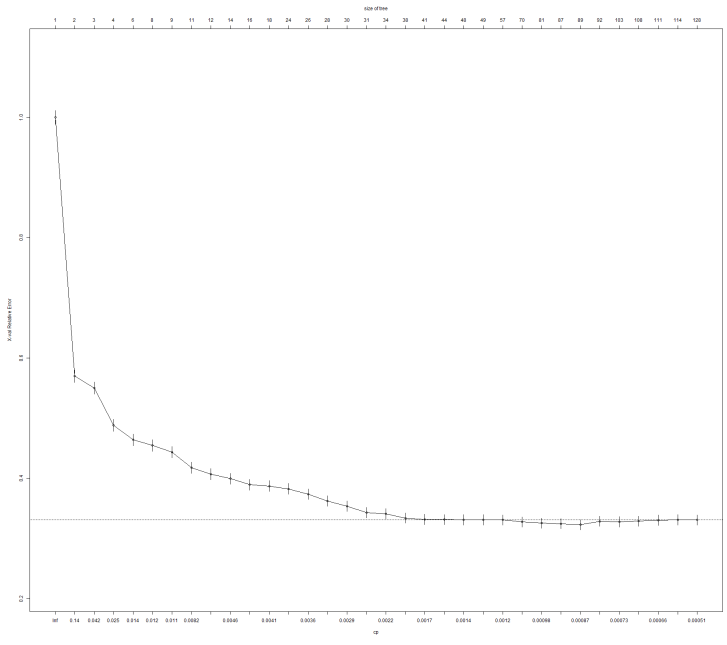
As a summary, the best model was xgboost with a cross-validation value of 1.000 and train model accuracy of 1.000. However, the best model implemented was the Two-Class Boosted Decision Tree in AzureML:

| **Model** | **Cross-Validation / Test accuracy** | **Implemented** |
|---|---|---|
| xgboost (Python) | 1.0000 | No |
| H2O ensemble (R) | 0.909 | No |
| NN model - Keras (Python) | 0.9085 | No |
| Two-Class Boosted Decision Tree (AzureML) | 0.9045 | Yes |
| Random forest (R) | 0.8975 | No |
| Decision tree (R) | 0.8200 | Yes |
| Nearest Neighbor (P) | 0.8111 | No |

## 3.1 Deployed models

### 3.1.1 Decision Tree (R)

We have used the model rpart that performs a recursive partition model for decision trees, the model by default use a 10-fold cross validation. The following is the relative error based on the size of the tree (note that the image shows the relative error, not the accuracy):



The model presents an accuracy of 0,8224 in cross-validation, with a training accuracy of 0.8941. We have tried to prune the tree based on the complexity parameter 0.0017 but the improvement was minimal, so we decided to use the first model calculated. The following is the classification of the training set:

| **Prediction\Real** | **Edible** | **Poisonous** |
|---|---|---|
| edible | 3876 | 528 |
| poisonous | 332 | 3388 |

### 3.1.2 Two-Class Boosted Decision Tree (AzureML)

Since we used AzureML studio for creating the API, we have tried different models already implemented on it, the one that gave us better results was the Two-Class Boosted Decision Tree. We have used the following parameters:

- Maximum number of leaves per tree: 20
- Minimum number of samples per leaf node: 10
- Learning rate: 0.2
- Number of trees constructed: 100

We applied 10-fold cross-validation with 0.9045 accuracy, the accuracy achieved in the trained model was 0.912.

## 3.2 Other models tried

### 3.2.1 Random Forest (R)

We have tried also a random forest model with the library randomForest, we obtained a cross validation value of 0.8953 with a model that includes all the variables:

| **Nº of variables** | **Accuracy error** |
|---|---|
| 22 | 0.1047514 |
| 11 | 0.1169375 |
| 6 | 0.1532496 |
| 3 | 0.2038405 |
| 1 | 0.2759724 |

The trained model accuracy is 0.8975. The following is the feature importance resulted from the 11 best variables of the model:

| **Variable** | **MeanDecreaseGini** |
|---|---|
| odor | 740,21685 |
| spore.print.color | 424,38858 |
| gill.color | 382,5468 |
| ring.type | 301,46116 |
| stalk.color.below.ring | 228,02398 |
| cap.color | 220,54046 |
| stalk.color.above.ring | 216,91008 |
| habitat | 180,2555 |
| stalk.surface.above.ring | 175,86914 |
| population | 175,14086 |
| stalk.root | 173,9069 |

We could not implement this model due to problems with input format data of the application.

### 3.2.2 H2O automl (R)

We also were interested in trying the functionality of H2O R library of automl, that tries different models (Gradient boosting, deep learning models among others and ensembles of all of them) and performs a comparison among all of them. For this model we have use the dataset converted with one hot encoding.

The results we obtained were the following:

| model_id | auc | logloss | meanerror | rmse |
|---|---|---|---|---|
| StackedEnsemble_AllModels | 0.9879655 | 0.1509320 | 0.05283648 | 0.2035472 |
| GBM_grid_0_AutoML_model_1 | 0.9876097 | 0.1432032 | 0.05452560 | 0.2043654 |
| GBM_grid_0_AutoML_model_4 | 0.9875130 | 0.1425388 | 0.05503543 | 0.2031066 |
| GBM_grid_0_AutoML_model_2 | 0.9870272 | 0.1477164 | 0.05713653 | 0.2073286 |
| GBM_grid_0_AutoML_model_0 | 0.9868853 | 0.1467957 | 0.05597269 | 0.2062910 |
| GBM_grid_0_AutoML_model_7 | 0.9867110 | 0.1484733 | 0.05622502 | 0.2067563 |
| GBM_grid_0_AutoML_model_3 | 0.9864590 | 0.1516443 | 0.05844972 | 0.2094648 |
| GBM_grid_0_AutoML_model_11 | 0.9863503 | 0.1519976 | 0.05486549 | 0.2090401 |
| GBM_grid_0_AutoML_model_8 | 0.9859543 | 0.1577695 | 0.05945907 | 0.2122935 |
| StackedEnsemble_BestOfFamily_0 | 0.9854392 | 0.1601496 | 0.05491699 | 0.2096911 |
| GBM_grid_0_AutoML_model_10 | 0.9840987 | 0.1733971 | 0.06127693 | 0.2206533 |
| GBM_grid_0_AutoML_model_13 | 0.9784133 | 0.2089236 | 0.07266819 | 0.2382878 |
| DRF_0_AutoML_20181215_134451 | 0.9736920 | 0.2529450 | 0.07964096 | 0.2648858 |
| GBM_grid_0_AutoML_model_9 | 0.9728678 | 0.6624175 | 0.08573827 | 0.4844166 |
| GBM_grid_0_AutoML_model_5 | 0.9720845 | 0.2258905 | 0.08425514 | 0.2527813 |
| GBM_grid_0_AutoML_model_14 | 0.9718884 | 0.2458850 | 0.08875603 | 0.2610675 |
| XRT_0_AutoML | 0.9712370 | 0.2589469 | 0.08528509 | 0.2692870 |
| GBM_grid_0_AutoML_model_15 | 0.9702524 | 0.5219401 | 0.08155667 | 0.4074411 |
| DeepLearning_0_AutoML_13 | 0.9580632 | 0.2835313 | 0.11208956 | 0.2889005 |
| GLM_grid_0_AutoML_model_0 | 0.9548884 | 0.2712512 | 0.11202777 | 0.2866848 |
| GBM_grid_0_AutoML_model_6 | 0.9466528 | 0.6773095 | 0.12414514 | 0.4920273 |
| GBM_grid_0_AutoML_model_12 | 0.7713079 | 0.6739473 | 0.21455424 | 0.4598691 |

As we can see the best model was the ensemble of all models that have an auc of 0.9879 and an accuracy of 0.909. This is its confusion matrix for the 80% of the data:

| | Edible | Poisonous | Error |
|---|---|---|---|
| Edible | 2699 | 407 | 0.131037 |
| Poisonous | 188 | 3187 | 0.055704 |
| Totals | 2887 | 3594 | 0.091807 |

We could not implement this model due to the fact that needs java to be executed and java was not available in the virtual machine.

#### 3.2.3 Keras model (Python)

In order to implement a neural network, we have used the keras library and we have set up a simple NN with the schema $126 : 60 : 1$.

The model had a 5-fold cross-validation accuracy of 0.9085 (60 epochs, batch-size of 100), with a trained model accuracy (20 epochs, batch size of 100) of 0.9200

We could not implement this model because we could not manage to use python in AzureML studio.

#### 3.2.4 Scickit model (Python)

We decided to use the nearest neighbors classifier provided by the scikit learn library, for this model divided the dataset in train and as explained in data preparation, obtaining a test score of 0.8111 and a train score of 0.8889.

We could not implement this model because we could not manage to use python in AzureML studio.

#### 3.2.5 xgboost model (Python)

Finally, we tried locally also a xgboost model. In this case we have extracted 1000 random records as test set, then we have spited the remaining records in test/train with sklearn train_test_split and we have applied the following parameters for the cross validation:

- 'eta': 0.02
- 'max_depth': 5
- 'subsample': 0.7
- 'colsample_bytree': 0.7
- 'objective': 'binary:logistic'
- 'seed': 99
- 'eval_metric':'error'
- 'nthread':4

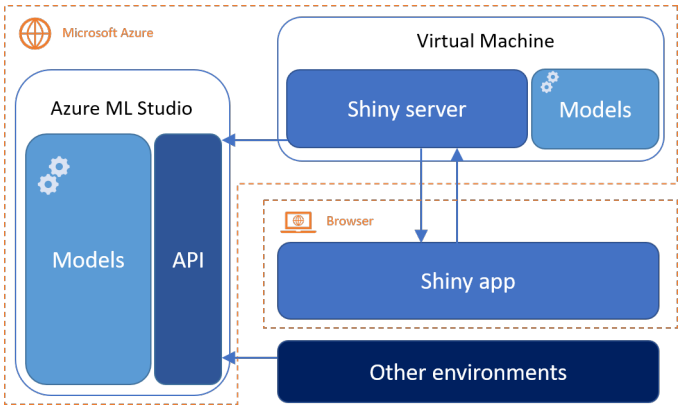
We have obtained 100% accuracy in the three cases the cross-validation, the model and the test set.

We could not implement this model because it was not possible to install the library in the virtual machine.

# 4 Deployment

## 4.1 Overview

The following figure shows a schema of the whole platform with its main components that have been developed. In the next sections, each of these components will be describe, as well as the usability of the platform and consumption of the models.




## 4.2 Infrastructure

For the development of this assignment, it has been used several technologies, frameworks and services hosted in Microsoft Azure. There are two main components deployed in this cloud platform.

On the one hand, we have used **Azure Machine Learning** studio in order to deploy different models and make them accessible on the part of third parties from other environments by consuming a public API. The use of this API is covered in the next section.

On the other hand, we have deployed a **virtual machine** with Ubuntu 14.04, installed the necessary packages and dependencies (such as R and shiny) and hosted a shiny server listening to the port 3838 containing our shiny app.

As we have developed the shiny app using GitHub as control version platform, the deployment of the application into the virtual machine, better said the shiny server, is quite straightforward using git commands once connected to the virtual machine through the ssh protocol. By using this approach, the deployment of the shiny app for its use is not necessary on the part of the users and can be reached publicly through the following link.

### 4.3 Usability and interface

Regarding the use of the platform and consumption of the models, we have provided two different ways.

The first one consists of the use of a shiny app in which the user can perform different actions. A brief explanation of the four parts of the application is given below.

- The **first tab** of the application is related to the prediction of the class of unknown mushrooms (e.g. edible or poisonous). In this tab, the user can either insert a custom mushroom to be predicted or upload a csv file containing the attributes of one mushroom per row. Immediately, the results of the predictions appear at the end of the page. It is important to note that this csv file cannot contain the attribute class as this is the attribute supposed to be predicted.

- The **second tab** of the application is focused on the exploration the set of mushrooms.

- In the **third tab**, a brief description of the API, input and output formats and some code examples are included. The main point here is the key value pairs on the top of the page, where the user can find the API key token and the endpoint of the resource where the model is located.

- Lastly, the **fourth tab** is reserved to present the members of the team behind the platform.

The second way of interacting with the platform and consuming the models to make predictions of unknown mushrooms is through a publicly available API built on top of Azure Machine Learning studio.

By this way, the access of the models from any environment able to make HTTP requests is guaranteed. There are two main concepts to explain here.

- **Security and access**, The models are located in the following endpoint through a POST request.

https://europewest.services.azureml.net/workspaces/9f6cbb787ae24af8bd2cb1fe82f331a7

/services/2916893a381f4fdcaeb283d15618340b/execute?api-version=2.0&details=true

To be allowed to consume the model, the client has to include the following bearer token to the header of the request.

stBIvZeMi7nCKQyDTKx2rC3CaCWzmyKgE7o0qh6jhcE5

rseLXxFhuqZd7hSPoI40KKV+80pIc/GtsF+Wqkzs7A==

- **Payload,** The data set of unknown mushrooms must be serialized into a JSON object with the following format, where the Values property contains one array of values per mushroom.

```
{
  "Inputs": {
    "input1": {
      "ColumnNames": [
        "cap-shape",
        "cap-surface",
        "cap-color",
        "bruises",
        "odor",
        "gill-attachment",
        "gill-spacing",
        "gill-size",
        "gill-color",
        "stalk-shape",
        "stalk-root",
        "stalk-surface-above-ring",
        "stalk-surface-below-ring",
        "stalk-color-above-ring",
        "stalk-color-below-ring",
        "veil-type",
        "veil-color",
        "ring-number",
        "ring-type",
        "spore-print-color",
        "population",
        "habitat"
      ],
```

```
        "Values": [
          [
            "value",
            "value",
            "value",
            "value",
            "value",
            "value",
            "value",
            "value",
            "value",
            "value",
            "value",
            "value",
            "value",
            "value",
            "value",
            "value",
            "value",
            "value",
            "value",
            "value",
            "value"
          ]
        ]
      }
    },
    "GlobalParameters": {}
}
```

In order the interpret the predictions, a response of the request with this format must be expected. The last two values in each array corresponds with the predicted class and the probability, respectively.

```
{
  "Results": {
    "output1": {
      "type": "DataTable",
      "value": {
        "ColumnNames": [
          "cap-shape",
          "cap-surface",
          "cap-color",
          "bruises",
          "odor",
          "gill-attachment",
          "gill-spacing",
          "gill-size",
          "gill-color",
          "stalk-shape",
          "stalk-root",
          "stalk-surface-above-ring",
          "stalk-surface-below-ring",
          "stalk-color-above-ring",
          "stalk-color-below-ring",
          "veil-type",
          "veil-color",
          "ring-number",
          "ring-type",
          "spore-print-color",
          "population",
          "habitat",
          "Scored Labels",
          "Scored Probabilities"
        ],
        "ColumnTypes": [
          "String",
          "String",
          "String",
          "String",
          "String",
          "String",
          "String",
          "String",
          "String",
          "String",
```

```
            "String",
            "String",
            "String",
            "String",
            "String",
            "String",
            "String",
            "String",
            "String",
            "String",
            "String",
            "String",
            "String",
            "Numeric"
        ],
                "Values": [
            [
              "value",
              "value",
              "value",
              "value",
              "value",
              "value",
              "value",
              "value",
              "value",
              "value",
              "value",
              "value",
              "value",
              "value",
              "value",
              "value",
              "value",
              "value",
              "value",
              "value",
              "value",
              "value",
              "value",
              "0"
            ]
          ]
        }
      }
    }
  }
}
```