Jupyter Notebooks Primer

- Jupyter notebooks allow you to run code in your browser
- In a nutshell, notebooks consists of cells, that are either markdown syntax (https://en.wikipedia.org/wiki/Markdown) (for comments etc.) or code
- Code cells are executed by selecting them and hitting "Shift-Enter" or clicking on Run above
- For more info have a look at this introduction (https://jupyter-notebook.readthedocs.io/en/stable/examples/Notebook/What%20is%20the%20Jupyter%20Note

The Input Data

The next cell loads pre-processed data for hundreds of samples. Columns include variant calls and some meta-information

In [1]:
```
import pandas as pd
# load the csv file as pandas dataframe
url = "https://raw.githubusercontent.com/andreas-wilm/microsoft-roadshow-hon
df = pd.read_csv(url)
```

In [2]:
```
# display the dataframe
df
```

13	sample-14	3	1	0	0	0	0	0	0	0	...	0	1
14	sample-15	1	0	0	0	0	1	0	0	0	...	0	0
15	sample-16	1	0	0	0	0	0	0	2	0	...	0	0
16	sample-17	1	0	0	0	0	0	0	0	0	...	0	0
17	sample-18	3	0	0	2	0	0	0	2	0	...	0	0
18	sample-19	2	0	0	0	0	0	0	0	0	...	0	0
19	sample-20	2	0	0	0	0	1	0	0	0	...	0	0
20	sample-21	1	0	0	0	0	0	0	0	0	...	1	0

In [3]:
```
# Remove columns that don't go into AutoML as features but keep a copy
annotation = df[["ID", "Status", "Gender"]].copy()
df = df.drop(["ID", "Status"], axis=1)
```

In [4]:
```
# Split into training and test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    df, annotation['Status'], test_size=0.2, random_state=42)
```

AutoML run (local)

A very cool feature in AutoML is automatic preprocessing (see `preprocess` below), which can automatically impute missing values, encode values, add features, embed words etc. See here (https://docs.microsoft.com/en-us/azure/machine-learning/service/how-to-create-portal-experiments#preprocess) for more information. Since the data-set here is clean already, there is no need for this.

To run AutoML we first define some basic settings

In [5]:
```
import logging

automl_settings = {
    "iteration_timeout_minutes": 1,
    "iterations": 10,
    "primary_metric": 'accuracy',
    "preprocess": False,
    "verbosity": logging.INFO,
    "n_cross_validations": 5
}
```

In [6]:
```
from azureml.train.automl import AutoMLConfig

automl_config = AutoMLConfig(task='classification',
                             debug_log='automated_ml_errors.log',
                             X=X_train.values,
                             y=y_train.values.flatten(),
                             **automl_settings)
```

```
WARNING - From /anaconda/envs/azureml_py36/lib/python3.6/site-packages/azur
eml/automl/core/_vendor/automl/client/core/common/tf_wrappers.py:36: The na
me tf.logging.set_verbosity is deprecated. Please use tf.compat.v1.logging.
set_verbosity instead.

WARNING - From /anaconda/envs/azureml_py36/lib/python3.6/site-packages/azur
eml/automl/core/_vendor/automl/client/core/common/tf_wrappers.py:36: The na
me tf.logging.ERROR is deprecated. Please use tf.compat.v1.logging.ERROR in
stead.
```

Connect to the ML workspace on Azure so that everything is logged there as well.

Please note: the following will require interactive authentication. Simply follow the instructions

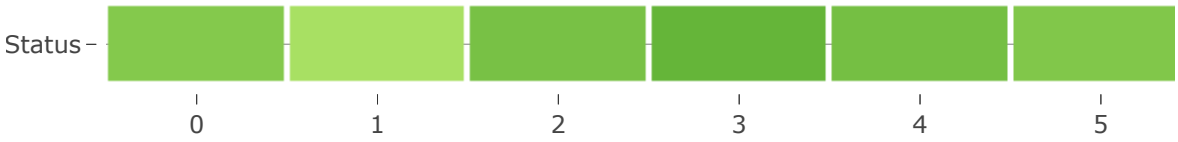
In [7]:
```
from azureml.core import Workspace
ws = Workspace.from_config()
```

In [8]:
```
# submit the experiment.
# note how automl runs multiple algorithms with different parameters automat
from azureml.core.experiment import Experiment
experiment = Experiment(ws, "vcf-classification-local")
local_run = experiment.submit(automl_config, show_output=True)
```

```
Running on local machine
Parent Run ID: AutoML_6672b35e-0db9-4d08-b4c6-2dd583762882
Current status: DatasetCrossValidationSplit. Generating CV splits.
Current status: ModelSelection. Beginning model selection.

********************************************************************************
************************
ITERATION: The iteration being evaluated.
PIPELINE: A summary description of the pipeline being evaluated.
DURATION: Time taken for the current iteration.
METRIC: The result of computing score on the fitted pipeline.
BEST: The best observed score thus far.
********************************************************************************
************************

 ITERATION   PIPELINE                                       DURATION      M
ETRIC      BEST
         0   StandardScalerWrapper SGD                     0:00:13
0.8625    0.8625
         1   StandardScalerWrapper SGD                     0:00:14
0.8050    0.8625
         2   MinMaxScaler SGD                              0:00:13
0.8825    0.8825
         3   MinMaxScaler RandomForest                     0:00:14
0.9125    0.9125
         4   StandardScalerWrapper RandomForest            0:00:15
0.8875    0.9125
         5   StandardScalerWrapper SGD                     0:00:13
0.8675    0.9125
         6   StandardScalerWrapper ExtremeRandomTrees      0:00:15
0.9175    0.9175
         7   MinMaxScaler RandomForest                     0:00:17
0.9350    0.9350
         8   VotingEnsemble                                0:00:13
0.9375    0.9375
         9   StackEnsemble                                 0:00:15
0.9250    0.9375
```

In [9]:
```
# Show the run details widget
from azureml.widgets import RunDetails
RunDetails(local_run).show()
```

AutoML_6672b35e-0db9-4d08-b4c6-2dd583762882:
Status: Completed

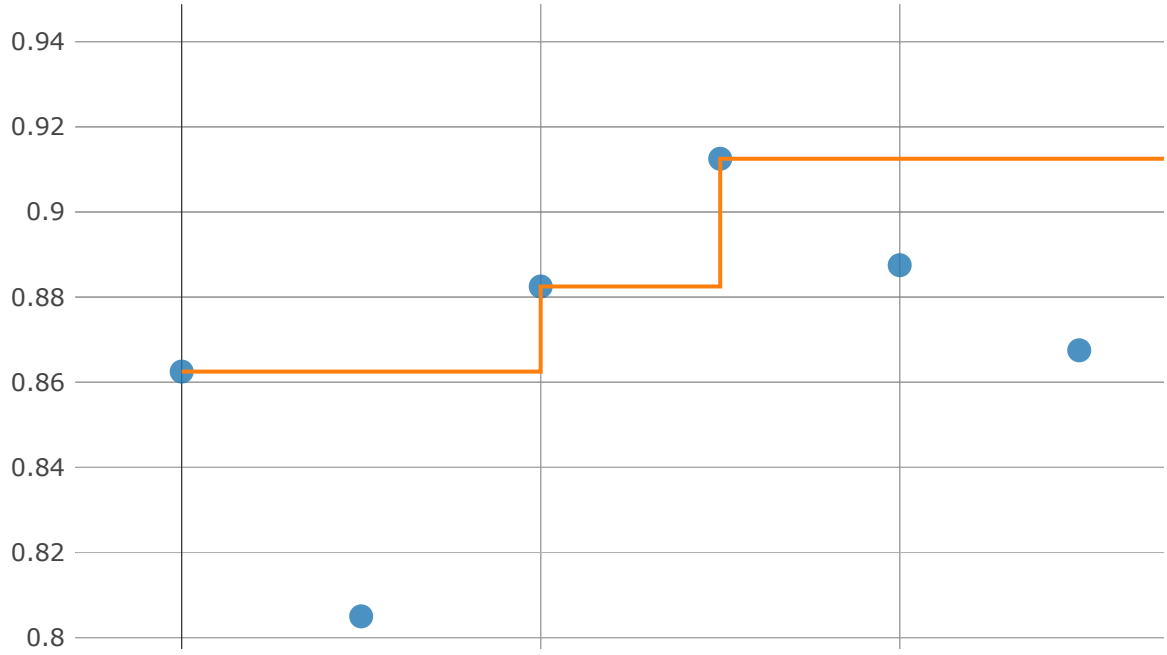


Iteration	Pipeline	Iteration metric	Best me
8	VotingEnsemble	0.9375	
7	MinMaxScaler, RandomForest	0.935	
9	StackEnsemble	0.925	
6	StandardScalerWrapper, ExtremeRandomTrees	0.9175	
3	MinMaxScaler, RandomForest	0.9125	

Pages: **1** 2 Next 5 per page
Last

accuracy

AutoML Run with metric : accu



Click here to see the run in Azure portal (https://mlworkspace.azure.ai/portal/subscriptions/0c331c07-92c0-4759-bb80-af51acabcbcc/resourceGroups/workshop/providers/Microsoft.MachineLearningServices/workspaces/classification-local/run/AutoML_6672b35e-0db9-4d08-b4c6-2dd583762882)

Predict outcome

In [10]:
```
# get the best model
best_run, fitted_model = local_run.get_output()
```

In [11]:
```
# predict outcome for 10 samples
y_predict = fitted_model.predict(X_test.values)
print("Sample\tPredicted\tActual")
for idx, (dfidx, dfrow) in enumerate(X_test.iterrows()):
    print("{}\t{}\t{}".format(annotation.at[dfidx, 'ID'],
                              y_predict[idx],
                              annotation.at[dfidx, 'Status']))
    # top 10 is enough
    if idx == 9:
        break
print("...")
```

```
Sample  Predicted       Actual
sample-362      1       1
sample-74       0       0
sample-375      0       0
sample-156      0       0
sample-105      1       1
sample-395      1       0
sample-378      0       0
sample-125      0       0
sample-69       1       1
sample-451      0       0
...
```

Print stats and plot a confusion Matrix

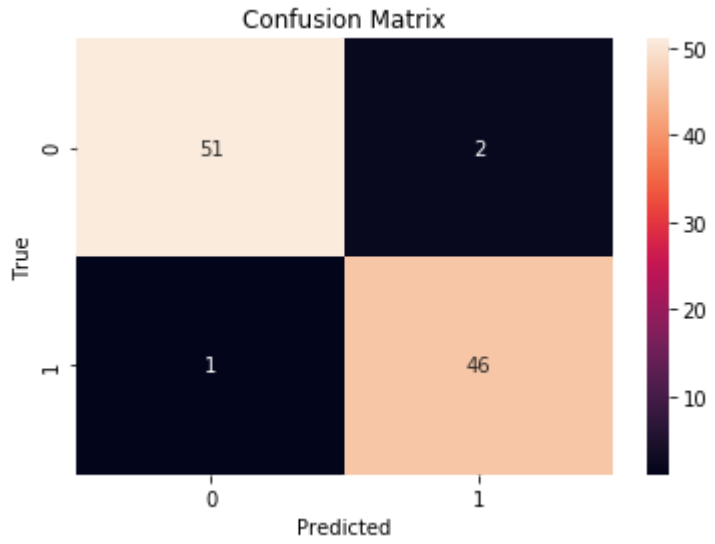
In [12]:
```
# idea from https://datatofish.com/confusion-matrix-python/
y_actual  = []
for dfidx, dfrow in X_test.iterrows():# what's the pandassy way of doing thi
    y_actual.append(annotation.at[dfidx, 'Status'])

data = {'y_Predicted': y_predict,
        'y_Actual': y_actual}
df = pd.DataFrame(data, columns=['y_Actual','y_Predicted'])
```

In [13]:
```
# print stats
from pandas_ml import ConfusionMatrix
Confusion_Matrix = ConfusionMatrix(df['y_Actual'], df['y_Predicted'])
Confusion_Matrix.print_stats()
```

```
population: 100
P: 47
N: 53
PositiveTest: 48
NegativeTest: 52
TP: 46
TN: 51
FP: 2
FN: 1
TPR: 0.9787234042553191
TNR: 0.9622641509433962
PPV: 0.9583333333333334
NPV: 0.9807692307692307
FPR: 0.03773584905660377
FDR: 0.04166666666666664
FNR: 0.02127659574468085
ACC: 0.97
F1_score: 0.968421052631579
MCC: 0.9400445871743088
informedness: 0.9409875551987152
markedness: 0.9391025641025641
prevalence: 0.47
LRP: 25.93617021276596
LRN: 0.022110972048393823
DOR: 1173.0000000000002
FOR: 0.019230769230769232
```

In [14]:
```
# plot confusion matrix
# idea from https://stackoverflow.com/questions/19233771/sklearn-plot-confus
import seaborn as sn

import matplotlib.pyplot as plt
ax = plt.subplot()

confusion_matrix = pd.crosstab(df['y_Actual'], df['y_Predicted'],
                               rownames=['Actual'], colnames=['Predicted'])

sn.heatmap(confusion_matrix, annot=True, ax = ax)

# labels, title and ticks
ax.set_xlabel('Predicted')
ax.set_ylabel('True')
ax.set_title('Confusion Matrix')
```

Out[14]: Text(0.5, 1, 'Confusion Matrix')



Model Interpretability and Explainability

Microsoft has six guiding AI principles (https://blogs.partner.microsoft.com/mpn/shared-responsibility-ai-2/). One of these is transparency, which states that it must be possible to understand how AI decisions were made. This is where model interpretability (https://docs.microsoft.com/en-us/azure/machine-learning/service/machine-learning-interpretability-explainability) comes into play. Here we will use a TabularExplainer to understand global behavior of our model.

In [15]:
```
from azureml.explain.model.tabular_explainer import TabularExplainer
# "features" and "classes" fields are optional. couldn't figure out how to u
explainer = TabularExplainer(fitted_model, X_train)
```

In [16]:
```
# Now run the explainer. This takes some time...
global_explanation = explainer.explain_global(X_train)
```

100% 400/400 [08:47<00:00, 1.32s/it]

In [17]:
```
# Let's find the top features
sorted_global_importance_names = global_explanation.get_ranked_global_names(
print("Top 10 features")
print("\n".join(sorted_global_importance_names[:10]))
```

```
Top 10 features
site-75
site-5
Gender
site-80
site-83
site-61
site-37
site-57
site-33
site-40
```

This should give you an idea about the causal factors in this data-set