

Advanced Machine Learning - Extend Operational Insights

Lab Guide

Overview

This lab guide contains the hands-on exercises for the **Advanced Machine Learning - Extend Operational Insights** workshop. Before proceeding with these exercises, please ensure that you have a copy of the workshop slide deck.

Download the workshop slide deck: <https://splk.it/AdvancedML-Attendee>

Prerequisites

In order to complete these exercises, you will need your own Splunk instance. Splunk's hands-on workshops are delivered via the [Splunk Show portal](#) and you will need a splunk.com account in order to access this.

If you don't already have a Splunk.com account, please create one [here](#) before proceeding with the rest of the workshop.

Troubleshooting Connectivity

If you experience connectivity issues with accessing either your workshop environment or the event page, please try the following troubleshooting steps. If you still experience issues please reach out to the team running your workshop.

- **Use Google Chrome** (if you're not already)
- If the event page (i.e. <https://show.splunk.com/event/<eventID>>) didn't load when you clicked on the link, try **refreshing the page**
- **Disconnect from VPN** (if you're using one)
- **Clear your browser cache and restart your browser** (if using Google Chrome, go to: Settings > Privacy and security > Clear browsing data)
- **Try using private browsing mode** (e.g. Incognito in Google Chrome) to rule out any cache issues
- **Try using another computer** such as your personal computer - all you need is a web browser! Cloud platforms like AWS can often be blocked on corporate laptops.

Table of Contents

Introduction	3
Before You Begin	3
Use Case 1: Detecting Potential Botnet Activity Using a Pre-Trained Model	7
Scenario	7
Steps	7
Use Case 2: Predicting Fraudulent Activity	9
Scenario	9
Model training and evaluation	9
Deploying an ONNX model to the Splunk Platform	11
(Optional Task) Sharing the model with other apps	13
Handy Notes for troubleshooting:	15
(Optional Task) Save model metadata and assess performance	16
Use Case 3: Threat Hunting Through Windows Event Logs	17
Scenario	17
Step 1 - Search for the relevant data	17
Step 2 - Create a dataset with meaningful features	17
Step 3 - Prepare our environment with the right data	18
Step 4 - Start exploring the data using the threat hunting notebook	20
Step 5 - Conclusion	20
(Optional) Use Case 4: Analyzing Network Traffic	21
Scenario	21
Steps	21

Introduction

In this workshop you will learn to create an ONNX model and upload the pre-trained ONNX model to Splunk's environment. We will use Splunk Machine Learning Toolkit (MLTK)'s latest feature introduced in version 5.4.0 which enables the option to upload pre-trained Open Neural Network Exchange (ONNX) models for inferencing in MLTK. You can train models in your preferred third-party environment, save the model in the .onnx file format, upload the model file to MLTK, and then retrieve and inference that model in MLTK. This option enables you to perform process-heavy model training outside of the Splunk platform, but benefit from model operationalization within the Splunk platform.

ONNX supports many ML libraries and offers language support for loading models and inference in Python, C++, and Java. To learn more, see <https://onnx.ai/>.

Throughout the workshop, we will be using python and Jupyter notebooks to create an ML model and the ML-SPL command `apply` to run it within MLTK. For information on these commands, see [Using the fit and apply commands](#).

In this workshop, you will train ML models on the same data for simplicity. **Please note that in realistic cases, the dataset should always be split into train and test data for more accurate performance evaluation.**

The following tools & datasets have been provided in your Splunk instance for this workshop:

- Splunk Machine Learning Toolkit (MLTK) app (v5.4.0)
- Python for Scientific Computing (PSC) Add-on
- Splunk App for Data Science and Deep Learning
 - Including a Jupyter Notebook dev environment
- Dataset:
 - Credit card fraud detection
 - BOTSv3

Your lab server is a cloud instance running on Amazon Web Services. You will need a web browser to connect to the server's Splunk web interface.

If you would like to run through this workshop in your own Splunk environment, please ensure you have the above listed apps and add-ons installed in your Splunk instance.

Before You Begin

Check that the dataset needed for this workshop is loading correctly.

1. Log in to your Splunk instance
2. Check your apps:
 - Go to **Apps > Manage Apps** section
 - Ensure the version of apps:

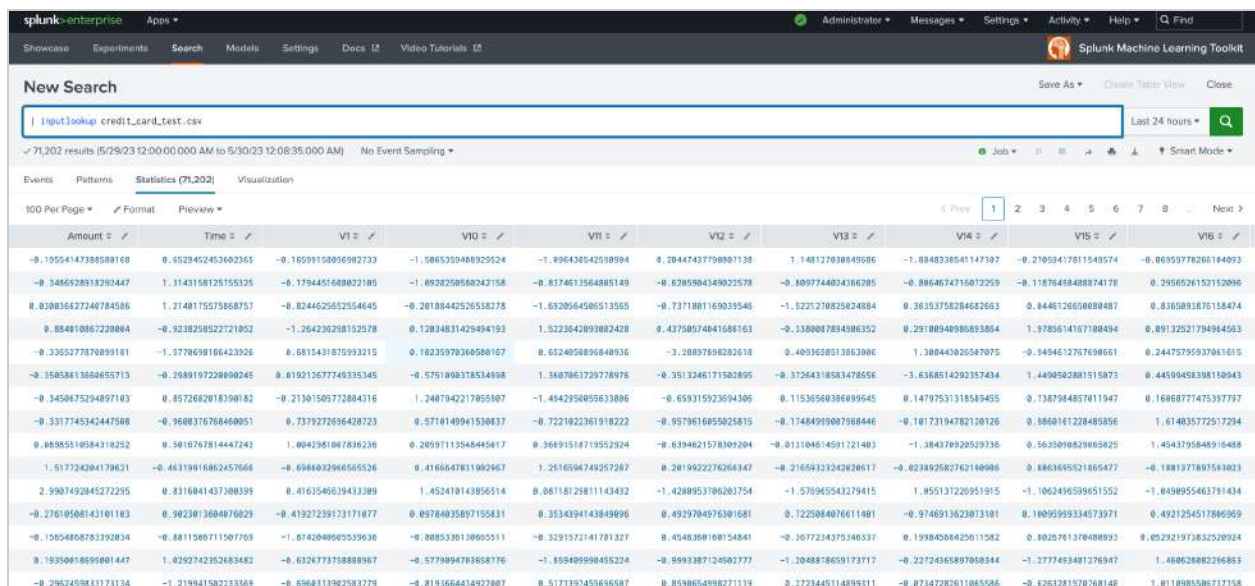
i. Splunk Machine Learning Toolkit = 5.4.0

ii. Python for Scientific Computing >= 4.1.0

- Ensure you also have Splunk App for Data Science and Deep Learning version 5.1.0

3. Check your data:

- a. Check that the **credit_card_test.csv** is present in your Splunk instance by navigating to **Settings > Lookups > Lookup Table Files**. The **credit_card_fraud.csv** lookup should be listed in the table of lookups.
- b. Check that the **BOTSv3** data is also present in your Splunk instance by navigating to **Settings > Indexes** and make sure **botsv3** is listed as one of your indexes.
- c. Run the following two searches over all time and make sure you see events:
 - i. `| inputlookup credit_card_test.csv`
 - ii. `index=botsv3`



Amount	Time	V1	V10	V11	V12	V13	V14	V15	V16
-0.1955414738858168	0.6529452453682365	-0.1659915895992733	-1.5865359488929524	-1.896438542599994	0.2844743779880138	1.148127838649586	-1.8848338541147387	-0.27059417811549374	-0.8695978268144893
-0.348628918292447	1.3143158125155325	-0.1794451688823185	-1.8924258568242158	-0.8374613564885140	-0.6395984349025578	-0.8097744804368285	-0.8864674716072259	-0.1187648488874178	0.2956526152152096
0.02883627248748586	1.2148175579568757	-0.824425652554645	-0.28188442526558278	-1.6328564585513565	-0.7371881160039548	-1.5221278825824884	0.38353758284682683	0.8448126658888487	0.838891876158474
0.884810867228884	-0.9238295822721802	-1.264239298152578	0.12834831429494193	1.5223624893882428	0.4375857484168163	-0.3380087894586352	0.2918894898853854	1.9785614167708494	0.89132527794964563
-0.3365277870899181	-1.5778698186423926	0.6815431870993215	0.1823597036058167	0.6524058895848936	-3.288978982820218	0.4093658513863886	1.388443820587075	-0.8494612767698661	0.24475795937861615
-0.3585813686855713	-0.2889197288989245	0.819212677448335345	-0.5751808378534498	1.3887861729778976	-0.3513246171580895	-0.37264318583478556	-3.638514292357434	1.4494582881515873	0.4459458388188943
-0.3458675294897183	0.8572682818398182	-0.21301506722884316	1.2487942217855807	-1.4842958855633886	-0.659315923694305	0.11536568386899645	0.14797531318589455	-0.738794857811947	0.16868777475397797
-0.3317745342447588	-0.9688376788469051	0.737272696428723	0.5718140941538837	-0.7221822361918222	-0.9578616855825815	-0.17464993807988446	-0.18173194782128126	0.8868161228485856	1.614835772577224
0.885510984318252	0.581676184447243	1.894381887836238	0.28597113548445817	0.3689151871952924	-0.6394621578395284	-0.911104614591121483	-1.384378928529736	0.563891852886820	1.4543795848915488
1.5177324284178621	-0.46319916862457666	-0.894889396655526	0.4166567831892967	1.2516594749257287	0.281982276266347	-0.21658323242828617	-0.823893582762188988	0.883655521865477	-0.1881377897583823
2.9907492845272295	0.8316841437388395	0.4161548639433389	1.452418143856514	0.88118125811143432	-1.428895378628754	-1.570945543279415	1.855137226951915	-1.106243659851552	-1.8498955463791434
-0.27019588143181183	0.9823813684876829	-0.41927239173171877	0.89788835897155831	0.3534394143849896	0.4929784976301681	0.7225884876611481	-0.9748913623873181	0.1809599334573971	0.4921254517886959
-0.13654886783392834	-0.8811986711587705	-1.8742848895359536	-0.8885336138865511	-0.3291572141781327	0.4548388168154841	-0.3677234375346537	0.19884588425611562	0.8029761378488893	0.652321973832528924
0.19350818699881447	1.8292742352683482	-0.6326773758888867	-0.5778894783658176	-1.858489984855224	-0.9993887124582777	-1.2048818659173717	-0.2272436589708344	-1.2777453481276947	1.488628882266853
-0.296245983173134	-1.219941582233388	-0.6968313982582779	-0.8193664434927807	-0.5171397455696587	0.858854988271119	0.2723445114893311	-0.87347282611845586	-0.8263281578768148	1.8118985586737158

Step 0: Image of a correctly loading credit_card_test dataset

4. In order to perform the onnx model upload on splunk, you need to ensure that you are enabled for the following capabilities:

- **upload_lookup_files**
- **upload_onnx_model_file**.

Alternatively, you can also use the **mltk_model_admin** role that includes the two above mentioned role permissions. You can validate assigned roles and capabilities on the **Users** page by navigating to **Settings > Users**. Make sure that the role **mltk_model_admin** has been assigned against your user name.



Name	Actions	Authentication system	Full name	Email address	Time zone	Default app	Default app inherited from	Roles	Last login	Status
admin	Edit	Splunk	Administrator	changeme@example.com	America/Los_Angeles	launcher	system	admin_mltk_model_admin	6/22/2023, 1:49:02 PM	Active

- Check that the Splunk App for Data Science and Deep Learning is set up and configured for use. Navigate to the **Splunk App for Data Science and Deep Learning**, then click on **Configuration > Containers**. Make sure the **1. Golden Show CPU** container image is running. If it is not running, select it from the dropdown menu and click **START**. Once running, browse to the **jupyter_url** that should be shown in the table below. Note that the Jupyter environment may take a few minutes to start up.

The screenshot shows the 'Containers' dashboard in Splunk. It includes a 'Development Container' section with dropdowns for 'Container Image' (set to '1. Golden Show CPU'), 'GPU runtime' (set to 'none'), and 'Cluster target' (set to 'docker'). Below these are buttons for 'START', 'STOP', and 'JUPYTER LAB'. To the right, a 'Status of all Container Models' section shows a bar chart with 'Active' at 1 and 'Inactive' at 18. At the bottom, a table lists container models with columns for model ID, mode, sharing, cluster, image, runtime, and various URLs.

model ID	mode	sharing	cluster	image	runtime	api_url	jupyter_url	mlflow_url	spark_url	tensorboard_url
1	DEV	global	docker	splunk-container-golden-cpu-5.1.0-show	none	https://localhost:4519	https://10.10.10.10:8888	https://10.10.10.10:8888	https://10.10.10.10:8888	https://10.10.10.10:8888
2		global								

- In the JupyterLab environment navigate to **/notebooks/** and make sure the following notebooks are present:

- **a_MLTK_ONNX_Workshop.ipynb**
- **A_threat_hunting_notebook.ipynb**
- **a_network_analysis.ipynb**

Also check in the **/notebooks/data/** directory that the **credit_card_fraud.csv** and **nn_botnet.onnx** files are present.

The screenshot shows the JupyterLab file browser interface. The left sidebar shows the file tree with the following structure:

- / notebooks /
 - data
 - libs
 - logs
 - a_MLTK_ONNX_workshop.ipynb
 - a_network_analysis.ipynb
 - a_threat_hunting_notebook_conf23.ipynb
 - a_threat_hunting_notebook.ipynb
 - anomaly_detection_ecod.ipynb

The right pane shows the 'Last Modified' column for each file, with values ranging from '20 hours ago' to '6 months ago'.

FileEditViewRunKernelGitTabsSettingsHelp

+

Filter files by name

/ notebooks / data /

Name	Last Modified
<div></div> credit_card_fraud.csv	8 days ago
<div></div> nn_botnet.onnx	8 days ago
<div></div> pretrained_dga_model_dSDL.json	5 months ago
<div></div> README	6 months ago

Use Case 1: Detecting Potential Botnet Activity Using a Pre-Trained Model

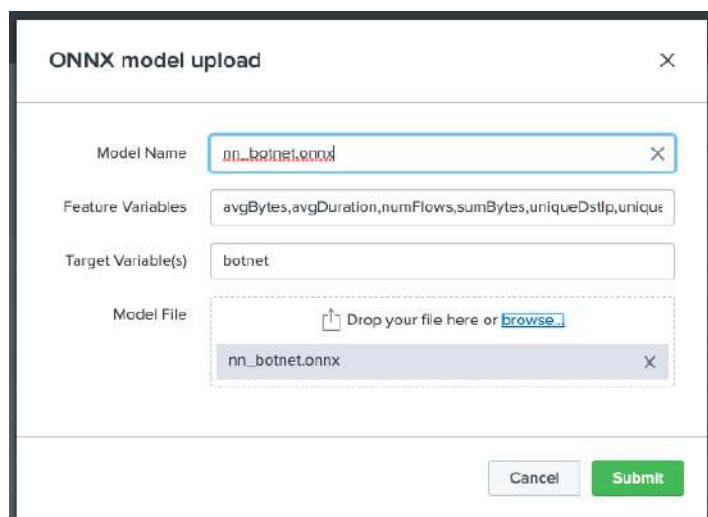
Scenario

At Buttercup Bank the security data science team have been researching if they can detect botnet activity based on netflow type logs. They have completed a successful experiment where a neural network classifier has been trained on historic netflow logs to predict the presence of a botnet and want to deploy this classifier to Splunk to generate notable events for the security operations centre. They have shared a model with you for running in Splunk and want you to deploy it into the Splunk instance for generating alerts

Steps

1. Check that your JupyterLab environment is available by navigating to the **Splunk App for Data Science and Deep Learning** and going to **Configuration > Containers**. You should see the `__dev__` container is up and running, and clicking on the `jupyter_url` in the table will launch the JupyterLab environment. Log on to this environment with the password “**Sp1unk4DeepLearning**”.
2. In the JupyterLab environment browse to **notebooks > data** and download the `nn_botnet.onnx` model.
3. Navigate back to Splunk and browse to the **Splunk Machine Learning Toolkit** app and go to the **Models** tab. Clicking on **Upload ONNX Model** should launch a modal like the one in the screenshot below. Enter in all the relevant information and submit to upload your ONNX model. In this case you need to input:
 - **Model name:** `nn_botnet.onnx`
 - **Feature Variables:**
`avgBytes,avgDuration,numFlows,sumBytes,uniqueDstIp,uniqueDstPort,uniqueProtocols`
 - **Target variable:** `botnet`
 - **Model file:** drop in or browse to the `nn_botnet.onnx` model saved using the code in the previous section

Provided the validation checks complete successfully you should receive a message saying the *file has been uploaded successfully*. Typical reasons for failed uploads are model sizes being too large, which can be modified by browsing to the **Settings** tab in the MLTK app and clicking on the **Edit Default Settings** button. Other reasons are the user uploading the file not having the correct permissions, which are described in the [Before You Begin](#) section above.



The screenshot shows a modal window titled "ONNX model upload" with a close button (X) in the top right corner. The modal contains four input fields: "Model Name" with the value "nn_botnet.onnx", "Feature Variables" with the value "avgBytes,avgDuration,numFlows,sumBytes,uniqueDstIp,unique", "Target Variable(s)" with the value "botnet", and "Model File" which has a file input area showing "nn_botnet.onnx" and a "browse" link. At the bottom of the modal are "Cancel" and "Submit" buttons.

- There are some netflow logs in the BOTSv3 index. Run the following search over all time to return data that is in the right format for the model:

```
index=botsv3 sourcetype="stream:tcp"
| bin _time span=1m
| stats count as numFlows avg(bytes_in) as avgBytes sum(bytes_in) as sumBytes
avg(client_rtt) as avgDuration dc(dest_ip) as uniqueDstIp dc(dest_port) as
uniqueDstPort dc(protocol_stack) as uniqueProtocols by _time src_ip
```

- Note that this search creates fields with the same names as the names we input as feature variables when uploading the model. We can now use our model by running the following search:

```
index=botsv3 sourcetype="stream:tcp"
| bin _time span=1m
| stats count as numFlows avg(bytes_in) as avgBytes sum(bytes_in) as sumBytes
avg(client_rtt) as avgDuration dc(dest_ip) as uniqueDstIp dc(dest_port) as
uniqueDstPort dc(protocol_stack) as uniqueProtocols by _time src_ip
| apply onnx:nn_botnet
```

New Search Save As Create Table View Close

index=botsv3 sourcetype="stream:tcp"
 | bin _time span=1m
 | stats count as numFlows avg(bytes_in) as avgBytes sum(bytes_in) as sumBytes
 avg(client_rtt) as avgDuration dc(dest_ip) as uniqueDstIp dc(dest_port) as uniqueDstPort dc(protocol_stack) as uniqueProtocols by _time src_ip
 | apply onnx:nn_botnet

81,081 events (20/06/2018 09:00:00.000 to 20/06/2018 15:00:00.000) No Event Sampling

Events Patterns Statistics (4,466) Visualization

20 Per Page Format Preview

_time	src_ip	numFlows	avgBytes	sumBytes	avgDuration	uniqueDstIp	uniqueDstPort	uniqueProtocols	botnet	predicted(botnet)
2018-06-20 09:01:00	172.31.38.181	3	378.0	1110.0	0.0	1	1	1	1	0.0
2018-06-20 09:02:00	172.16.0.127	1	2753.0	2753.0	12.0	1	1	1	1	1.0
2018-06-20 09:02:00	172.31.12.76	1	2001.0	2001.0	8.0	1	1	1	1	1.0
2018-06-20 09:02:00	172.31.38.181	3	378.0	1110.0	0.0	1	1	1	1	0.0
2018-06-20 09:02:00	192.168.105.215	7	1652.0	11564.0	34.14205714285714	3	1	3	3	0.0
2018-06-20 09:03:00	172.31.38.181	3	378.0	1110.0	0.0	1	1	1	1	0.0
2018-06-20 09:03:00	192.168.105.215	11	21331.363836364	236045.0	84.27272727272727	9	1	9	9	0.0

- Note that the **predicted(botnet)** field should now be present with our prediction in it. We can filter our results to only show the predicted botnet traffic by running the following search:

```
index=botsv3 sourcetype="stream:tcp"
| bin _time span=1m
| stats count as numFlows avg(bytes_in) as avgBytes sum(bytes_in) as sumBytes
avg(client_rtt) as avgDuration dc(dest_ip) as uniqueDstIp dc(dest_port) as
uniqueDstPort dc(protocol_stack) as uniqueProtocols by _time src_ip
| apply onnx:nn_botnet
| where 'predicted(botnet)'>=1
```

- This search can now be saved as an alert to generate notable events every time a botnet is predicted.

Use Case 2: Predicting Fraudulent Activity

Scenario

You are employed as a Data Scientist in Buttercup Banks, which is one of the leading financial firms in the city. To beat the competition among several other financial services gaining popularity in the city, your firms' strategic advisory team has decided to introduce loyalty rewards for the customers holding a credit card with Buttercup Banks. As a consequence of this idea, the company created a marketing team dedicated to reaching out to valued customers who have been using these credit cards. However, as a part of this process, we also need to filter out candidates who are potentially inclined towards fraudulent activity. You have been assigned the task to develop a machine learning model that detects fraudulent transactions and triggers alerting based on the transactional data of customers. Identifying such triggers would also help the bank put checks in proper places and reduce the number of fraud occurrences for the firm.

To maintain customer confidentiality, you will not be provided with original features or other background information. Instead you are given a dataset that is already pre-processed with feature variables transformed into PCA components. The only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction amount and can be used for example-dependent cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

Ref:

<https://www.openml.org/search?type=data&id=42175&sort=runs&status=active>

<https://api.openml.org/api/v1/json/data/42175>

<https://api.openml.org/d/42175>

Model training and evaluation

8. Check that your JupyterLab environment is available by navigating to the **Splunk App for Data Science and Deep Learning** and going to **Configuration > Containers**. You should see the `__dev__` container is up and running, and clicking on the `jupyter_url` in the table will launch the JupyterLab environment. Log on to this environment with the password “**Sp1unk4DeepLearning**”.
9. Navigate to the **JupyterLab environment > notebooks** and open the **a_MLTK_ONNX_Workshop.ipynb** notebook.
10. To get started, first read the csv file. This can be done by running the first two cells in the notebook: “**Verify Package Installation and Imports**” and “**Read the dataset as pandas DataFrame**”. The target variable, “Class”, will either hold a value of 1 in case of fraud and 0 otherwise. All other variables are a part of the “feature” variables.

[3]:	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.20E

5 rows x 31 columns

11. Create train and test data by running the next code block. Note that the feature and target variables will be extracted during this section of code, with the data also being scaled using some numpy functions.

```

Create training/test data

[4]: # select predictor variables
features = [c for c in df.columns if "Class" not in c]
target = "Class"

# split data and train model with 75% of data
X_train, X_test, y_train, y_test = train_test_split(
    df[features], df[target], random_state=0
)

# Standard scale
mean, std = np.mean(X_train, axis=0), np.std(X_train, axis=0)
X_train = (X_train - mean) / std
X_test = (X_test - mean) / std

```

12. Lets create a Sequential model by stacking various layers using the Sequential class in `tf.keras` module and look at the model's architecture using `model.summary()`, which should result in a summary like the screenshot below. This can be done by executing the "Model creation and configuration" code block.

Model: "sequential"		
Layer (type)	Output Shape	Param #
dense (Dense)	(None, 256)	7936
dense_1 (Dense)	(None, 256)	65792
dropout (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 256)	65792
dropout_1 (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 1)	257
Total params: 139,777		
Trainable params: 139,777		
Non-trainable params: 0		
None		

13. Before training the model we want to define metrics that will be used to assess the model performance during the training process. These metrics can be defined and then captured using `model.compile`. Once we have run the model compilation code block, the subsequent section of code will `fit` the model on the training dataset.

```

Model Training

And just to test out the model, let's go ahead and define metrics to compile the model using model.compile and then run fit on the training dataset.

[6]: # model settings
metrics = [
    tf.keras.metrics.FalseNegatives(name="fn"),
    tf.keras.metrics.FalsePositives(name="fp"),
    tf.keras.metrics.TrueNegatives(name="tn"),
    tf.keras.metrics.TruePositives(name="tp"),
    tf.keras.metrics.Precision(name="precision"),
    tf.keras.metrics.Recall(name="recall"),
]

model.compile(
    optimizer=tf.keras.optimizers.Adam(1e-2), loss="binary_crossentropy", metrics=metrics
)

# use class weight due to highly imbalanced dataset
class_weight = dict(zip([0, 1], 1 / np.bincount(y_train)))

[7]: # model training
model.fit(
    X_train,
    y_train,
    batch_size=2048,
    epochs=50,
    validation_data=(X_test, y_test),
    class_weight=class_weight,
)

103/105 [=====] - 1s 11ms/step - loss: 3.0801e-07 - fn: 2.0000 - fp: 2498.0000 - tn: 210735.0000 - tp: 370.0000 - precision: 0.0553 - recall: 0.9812 - val_loss: 0.0518 - val_fn: 12.0000 - val_fp: 1372.0000 - val_tn: 69710.0000 - val_tp: 108.0000 - val_precision: 0.0730 - val_recall: 0.9000
Epoch 47/50
105/105 [=====] - 1s 11ms/step - loss: 3.0801e-07 - fn: 2.0000 - fp: 2498.0000 - tn: 210735.0000 - tp: 370.0000 - precision: 0.1290 - recall: 0.9946 - val_loss: 0.0332 - val_fn: 12.0000 - val_fp: 666.0000 - val_tn: 70416.0000 - val_tp: 108.0000 - val_precision: 0.1395 - val_recall: 0.9000
Epoch 48/50
105/105 [=====] - 1s 11ms/step - loss: 3.1879e-07 - fn: 4.0000 - fp: 2331.0000 - tn: 210902.0000 - tp: 368.0000 - precision: 0.1395 - val_recall: 0.9000

```

- To evaluate our model we are going to run the “**Model Evaluation**” code block, which will calculate the evaluation metrics defined above against the test datasets. Provided you are satisfied with the evaluation results, you can save the tensorflow model. Note that you may not see identical results to those presented in the notebook (87.5% recall and 0.7% false positive rate), but should see good accuracy from this model.
- In preparation for deploying the model using MLTK we are now going to save the model in an ONNX format. In this step, you will learn how to convert your tensorflow model to a common file format called Open Neural Network Exchange (ONNX). ONNX supports many ML libraries and offers language support for loading models and inference in Python, C++, and Java. To learn more, see <https://onnx.ai/>. The “**Saving the model**” code block allows us to do this, and once executed should save a model called **credit_card_fraud.onnx** in the **/notebook/data/** folder in the JupyterLab environment. Once you have located the model file, download it - you’ll be needing it for subsequent tasks!

```

Saving the model

[ ]: import tf2onnx
import onnx

onnx_model, _ = tf2onnx.convert.from_keras(model, opset=13)
onnx.save(onnx_model, 'data/credit_card_fraud.onnx')

#features: Time, V1, V2, V3, V4, V5, V6, V7, V8, V9, V10, V11, V12, V13, V14, V15, V16, V17, V18, V19, V20, V21, V22, V23, V24, V25, V26, V

```

Deploying an ONNX model to the Splunk Platform

Congratulations! You presented your work to the whole team and they seem to be pretty impressed with the results. However, you’ve used a non-splunk based env (JupyterLab) to create your model and the company uses Splunk to analyze real time transactions. To merge into the company’s typical workflows, the team wants you to import your model into Splunk’s environment to enable real-time fraud detection. Here comes your next challenge: create a strategy to deploy your model to Splunk.

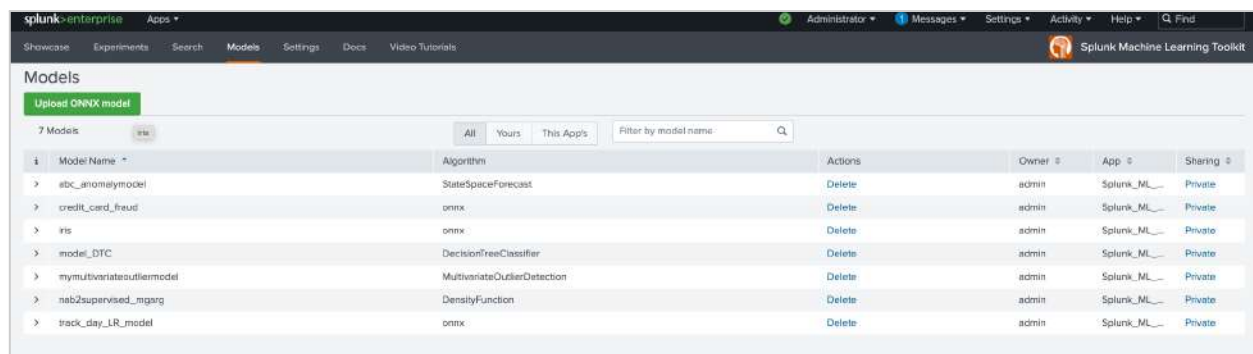
We will then use MLTK to upload your pre-trained ONNX models for inferencing. This capability enables you to perform process-heavy model training outside of the Splunk platform while benefiting from model operationalization in the Splunk platform.

Reminder: In order to perform the next steps of uploading ONNX model on splunk, you need to ensure that you are enabled for the following capabilities:

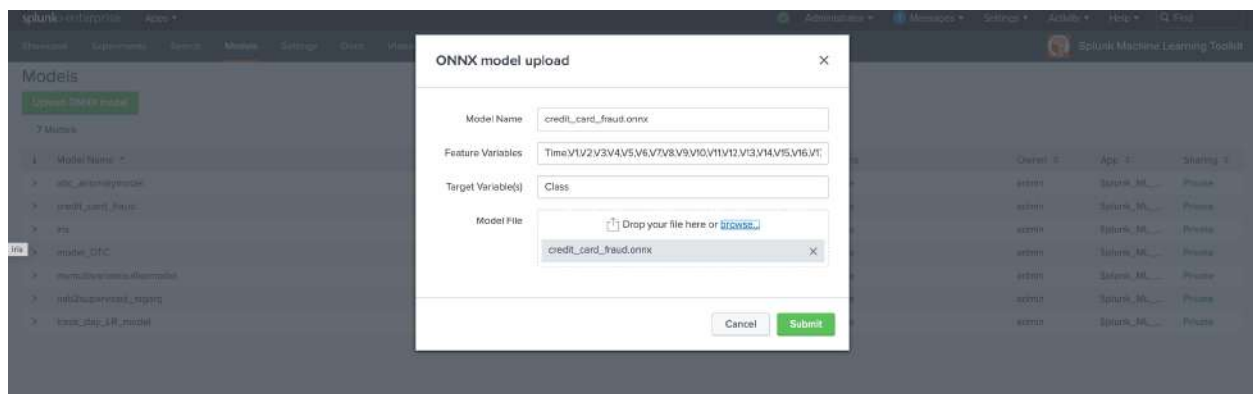
- **upload_lookup_files**
- **upload_onnx_model_file**

Alternatively, you can use the **mltk_model_admin** role that includes the two required role permissions and ships with MLTK. For more information, check pt.1 in the '**Handy Notes for troubleshooting**' section at the bottom of this exercise.

1. Open MLTK. Make sure the version is 5.4.0 (and above). Click on the “**Models**” tab.
2. Once the **Models** page loads, click on the “**Upload ONNX Model**” button.



3. It opens up a dialog box which expects you to enter the model settings:



4. Enter the model name, features, target variable used for the model file, and upload the onnx model file. The feature names can be found in the “**Saving the model**” code block in the JupyterLab for copying and pasting in. Once this information has been input click on the **Submit** button.

- A message will notify you that your model has been uploaded successfully, but the Models page will need to be refreshed to see the model in the listing. Once you have confirmed the model has successfully uploaded, go to the **Search** tab and use the following query to infer your model:

```
| inputlookup credit_card_test.csv
| apply onnx:credit_card_fraud
| table V* predicted(Class)
```

The screenshot shows the Splunk Enterprise Search interface. The search query is: `| inputlookup credit_card_test.csv | apply onnx:credit_card_fraud | table V* predicted(Class)`. The results are displayed in a table with columns V27 through V99 and a final column for predicted(Class). The predicted values are either 0 or 1, indicating the model's output for each data point.

	V27	V28	V3	V4	V5	V6	V7	V8	V9	predicted(Class)
104	0.278756373563834	0.4578161743916444	-0.83115644885900825	-0.428952247058986	0.3212548721884	-0.86674254847884435	0.9453064822148216	-0.1812557675531423	-0.158512063281728	0.012981306
586	0.19284145577494588	0.5424170780549445	0.8958186408164899	-0.4641234925344812	0.854784547094667	-0.5520618075496425	0.8237918013332227	-0.857227619847824975	-0.2742987563383862	1.4981161e-07
553	0.712728222298956	1.3515178438184196	0.2158159671231168	0.4687026262269593	1.731742202085912	-1.3369911876923137	-0.9274463884334172	0.2742357967914757	0.8211120525628586	-5.5984645e-08
805	-1.2658916150887494	-2.04195482592168637	0.9514812718343728	0.740564630944348	-1.3554158155010425	1.566812616281626	0.41703102767624586	-0.19968383378625426	-0.2366964588613986	-5.5984645e-08
313	-0.805834457834687923	0.88986689246523873	0.488962132576472	-0.158484014822811	-0.8188112486145156	-0.3956589177035376	-0.6523884714574485	-0.85111317612858118	0.9684122465173344	-5.5984645e-08
689	0.4869886434329208	-0.4587499770646463	-1.3264858947878905	1.1652582841321222	0.4589159479423031	-1.3136429419085364	0.5748207573688833	0.84875948309727541	-0.812065246671987576	0.821487e-06
453	0.9486119229434128	0.6326738576543635	1.8768891769206012	3.177523532985238	-0.27271766781126466	0.6791383737864267	-0.3298987818654972	0.5911808915221811	-1.3481838523388386	-5.5984645e-08
508	0.81835538918842097	0.848651484991736835	0.2289572628898978	-0.5561865326831342	-0.4519283996963853	-0.563962358114512	-0.4819283996963853	-0.18932626394358012	-0.6081641388459116	-5.5984645e-08
397	0.81615174884386663	-0.64835120451483785	-1.865543488388432	-1.1784427459032445	1.266725585664796	2.476912349338793	-0.9047231694986384	0.6785123948896867	-0.3851438919949457	-5.5984645e-08
984	0.64245785105849	0.256758495848943	1.2345743176293889	2.1514464412343957	-1.8588362454824417	1.8155673074544388	0.9415727864958866	0.288288885125714	-0.9466372351552168	1.8838485e-05
938	-0.6895929360838354	0.13368034820863186	-2.3855698428911825	-0.8888186148412864	-0.4229834893839617	-0.5258465869693739	0.7284965484878623	-0.4759555100843307	-1.364561926318888	-5.5984645e-08
644	-0.8913113578368005	0.82348867882345657	-0.9383218695885818	-0.4474412849226546	0.751615833952785	-0.2674351899850568	0.633846129854228	-0.811661492122954389	0.1611868185843584	-5.5984645e-08
582	-0.9831983257073347	-2.252937966118744	1.8285138317116689	0.5071880760725911	-0.97113751458974846	-0.89814172033495372	-0.571391848913928	0.7185288814273855	0.2481422816438381	-5.5984645e-08
849	-0.6962281918389861	-0.8126874828334449	0.3156812556488594	-1.821880986998792	-1.234280838537793	0.1258152824858986	-0.7482377783086771	0.8538149308953458	-0.180220387146859	-5.5984645e-08
631	0.2481163215158031	0.83464250419803187	1.3574727638879664	-0.6913648123857283	-0.4771224681428137	-0.85158806411193976	-0.280567328789788	0.624163255481297	-0.8847195828956687	-5.5984645e-08
636	0.811184646524619636	0.10579454638822836	0.27365578991926706	0.8141448791318745	-0.31794881338714584	-0.659948882681217	0.13842686963947646	-0.15897238522363852	0.8591816652835458	5.5984645e-08
643	0.8248358974858444	-0.14359437268613154	-0.454491341524839	-1.0184163339804995	-1.1955596344423887	-0.6622275991095884	-1.013635668374732	-0.11343761863726433	-0.7799763798135628	-5.5984645e-08
629	0.3128921925944936	0.6156479861983289	-0.2964973939225208	-0.934882719029436	0.352753264212184	0.194848345484085984	0.4370229393823595	-0.74493135221700967	-0.2842469936496576	-5.5984645e-08

(Optional Task) Sharing the model with other apps

If you are at this step, it means that you have been able to successfully deploy your model to splunk and make inference from the test data from within MLTK. Give yourself a good pat on the back! :) Coming back to our initial scenario, if you recall, we mentioned that your solution helps the firm put checks in place to mitigate fraudulent actions in the future. As a result, you are required to share your model with the IT division who are working to build a solution to track behaviour of the accounts labelled as fraudulent by your solution.

In this step, you will learn how to share your model with other apps to achieve valuable use cases.

- Navigate back to the **Models** tab and find your **credit_card_fraud** model file. Under the “**Sharing**” column click on the default shared value (e.g. “Private”) in the screenshot below to open the permissions configurations.

The screenshot shows the Splunk Models page. The model **credit_card_fraud** is listed. The 'Sharing' column shows 'Private'. The 'Permissions' section below the table indicates the model is owned by 'admin' and is 'Private'. The 'Algorithm' is 'onnx'. The 'Target Variable(s)' is 'Class'. The 'Feature Variables' are 'Time,V1,V2,V3,V4,V5,V6,V7,V8,V9,V10,V11,V12,V13,V14,V15,V16,V17,V18,V19,V20,V21,V22,V23,V24,V25,V26,V27,V28,Amount'.

Model Name	Algorithm	Actions	Owner	App	Sharing
credit_card_fraud	onnx	Delete	admin	Splunk_ML_Toolkit	Private

- If you want to use the ONNX model under the app namespace, you must change the model permissions to App or Global. From the “**Edit permissions**” dialog change the “**display for**” option to “**App**” from “**Owner**”.

Model Name

credit_card_fraud

Owner

admin

App

Splunk_ML_Toolkit

Display For

Owner

App

All apps

	Read	Write
Everyone	<input checked="" type="checkbox"/>	<input type="checkbox"/>
admin	<input type="checkbox"/>	<input type="checkbox"/>
can_delete	<input type="checkbox"/>	<input type="checkbox"/>
mltk_model_admin	<input type="checkbox"/>	<input type="checkbox"/>
power	<input type="checkbox"/>	<input type="checkbox"/>
psc_admin	<input type="checkbox"/>	<input type="checkbox"/>
splunk-system-role	<input type="checkbox"/>	<input type="checkbox"/>
user	<input type="checkbox"/>	<input type="checkbox"/>

Cancel

Save

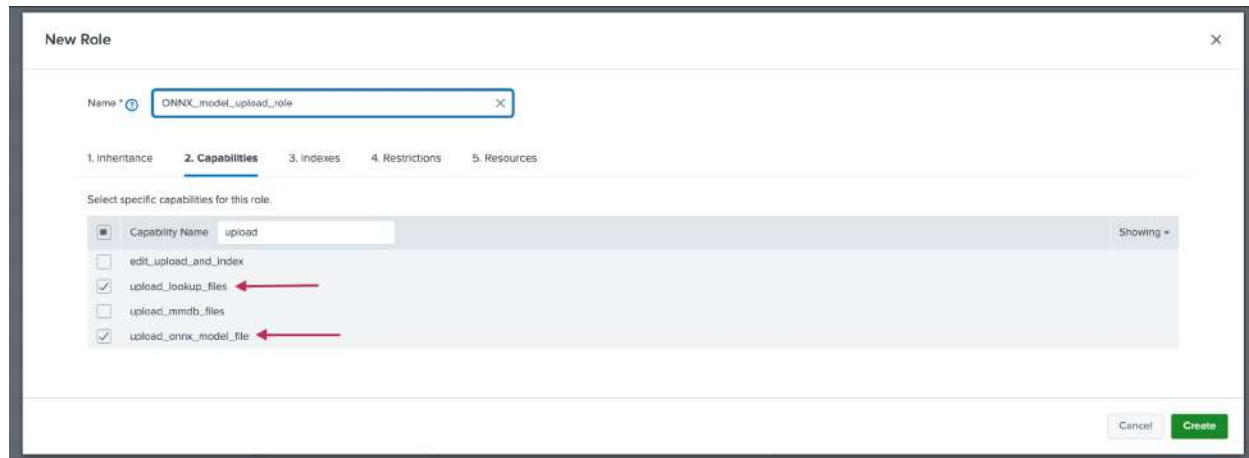
- In the search you must append the ONNX model call with the app keyword as shown in the following example.

```
| inputlookup credit_card_test.csv
| apply app:onnx:credit_card_fraud
| table V* predicted(Class)
```

New Search									
inputlookup credit_card_test.csv apply app:onnx:credit_card_fraud table V* predicted(Class)									
71,202 results (6/19/23 1:00:00.000 PM to 6/20/23 1:37:00.000 PM) No Event Sampling									
Events	Patterns	Statistics (71,202)	Visualization						
100 Per Page	Format	Preview							
V1	V10	V11	V12	V13	V14	V15	V16	V17	V18
-0.1659915889592733	-1.586535488029524	-1.896438542590404	0.28447437790887135	1.148127034849686	-1.8848338541147187	-0.2785941781154957	-0.80959778266184893	0.5951296885651298	0.4515588561062568
-0.1794451688822185	-1.8528258580242158	-0.8374613564805149	-0.6285984349022578	-0.8897744824366285	-0.8064674716872259	-0.11876498488874118	0.2956526152152895	0.5784791825787447	-0.17679789271860434
-0.8244625652554645	-0.2818844252653828	-1.6528564586513505	-0.7371801169019546	-1.5221278825824864	0.1635375828488265	0.04461266598888487	0.8365893876158474	-1.33589687327171328	0.9755256574178718
-1.2642326281525779	0.12934831429494193	1.5223642893882428	0.4375051404168616	-0.3388867854906352	0.29180949586893864	1.9785614167188491	0.89132521794964564	0.13686818909074254	0.3489288361504865
0.6815431875953215	0.18235970368588167	0.6524858896648936	-3.2889789828261886	0.4893658513863886	1.388443026587875	-0.549461276769866	0.2447579593786161	2.2873147485499296	-2.2757388878029254
0.019212677749335345	-0.5751888378534998	1.3687863729778976	-0.3513248171542895	-0.37264318581478656	-3.636851429235744	1.4498982881515873	0.44599458398158943	3.9377973186205963	1.8548789661125691
-0.2138150577288432	1.2487942217855987	-1.4942958855633886	-0.65931592364386	0.11536568368899645	0.14797531318585455	0.7387984857811947	0.18868777475397794	0.3201226854825503	0.6964729742897888
0.7379272696428723	0.5718149941538837	-0.7221822351918222	-0.9579616855825816	-0.17484999807968444	-0.18173134782128126	0.9880161228485856	1.614895772517294	0.85631984382656302	-1.6116841848527278
1.604281887830236	0.28597113548445815	0.3669151871955293	-0.6394621578349284	-0.813184614591721485	-1.384376928529736	0.563996829865825	1.4543795848916488	1.3837835184273645	-1.25432436161818
-0.6988326664283182967	0.416664283182967	1.7516556768757787	0.7819822736366347	-0.7185537324289814	-0.873852587367198881	0.8863885531865477	-0.18813728879583873	0.3687156924654867	0.17188362666544481

Handy Notes for troubleshooting:

1. **ONNX file upload permissions:** By default the ONNX model upload capability is turned off for all users. Your Splunk admin can turn on the upload capability by creating a custom role and assigning that role the following two required permissions:
 - **upload_lookup_files**
 - **upload_onnx_model_file**



Your Splunk admin can also add these permissions to any current role in order for that role to upload ONNX models.

To learn more see, [Set permission granularity with custom roles](#) in the *Securing Splunk Enterprise* manual.

As an alternative, you can use the **mltk_model_admin** role that ships with MLTK. This role already includes the two required role permissions.

2. **Use the correct ONNX opset value:** Opset values are important while converting models from sklearn, TensorFlow, or PyTorch to the ONNX format. You must use the correct opset value when creating your models in order to successfully upload your models. Incorrect opset values will lead to inference errors. Use the latest target opset value officially supported by onnx.ai.

To learn more, see

http://onnx.ai/sklearn-onnx/auto_tutorial/plot_cbegin_opset.html#what-is-the-opset-number.

3. **About ONNX model sizes:** ONNX models are natively binary files. These model files are converted to base64 encoded entries to create the lookup entry. This type of encoding increases the file size by approximately 33% to 37%. Take this into consideration when setting your model upload file size limitations.

There are limits on the maximum model file upload size. The maximum model file upload size is based on the size limitations defined in the `max_model_size_mb` setting in the `mlspl.conf` file. To learn how to change these settings, see [Edit default settings in the mlspl.conf file](#).

For more information on Troubleshooting ONNX Models check our official docs page for reference: [Troubleshoot ONNX model uploads](#)

(Optional Task) Save model metadata and assess performance

There are some optional code blocks at the bottom of the Jupyter Notebook for:

1. Saving the tensorflow model and it's assets
2. Saving the train and test datasets
3. Running inference using the ONNX model
4. Calculating some additional accuracy metrics using the model

1. You are able to view the statistics of this example with the following search over all time:

```
index=botsv3 source="*WinEventLog:Security"
| bin _time span=1h
| fillnull value=unknown Account_Name
| stats count by _time ComputerName Account_Name EventCode
```

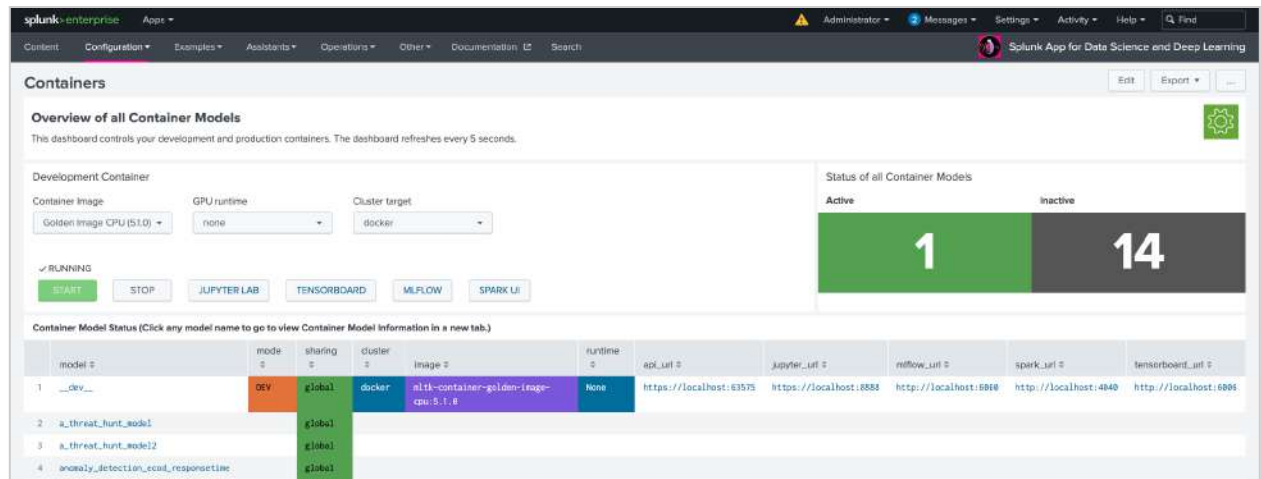
3. You scan the data and keep an eye out for any missing values. As you recall from your data science days, if there are missing values, it is pertinent to clean them.

_time	ComputerName	Account_Name	EventCode	count
2018-08-28 11:00	ABUNGST-L.froth.1j	-	4688	319
2018-08-28 11:00	ABUNGST-L.froth.1j	ABUNGST-LS	4616	1
2018-08-28 11:00	ABUNGST-L.froth.1j	ABUNGST-LS	4624	4
2018-08-28 11:00	ABUNGST-L.froth.1j	ABUNGST-LS	4627	4
2018-08-28 11:00	ABUNGST-L.froth.1j	ABUNGST-LS	4663	1
2018-08-28 11:00	ABUNGST-L.froth.1j	ABUNGST-LS	4678	42
2018-08-28 11:00	ABUNGST-L.froth.1j	ABUNGST-LS	4673	67
2018-08-28 11:00	ABUNGST-L.froth.1j	ABUNGST-LS	4688	355
2018-08-28 11:00	ABUNGST-L.froth.1j	ABUNGST-LS	4689	389
2018-08-28 11:00	ABUNGST-L.froth.1j	AlBungstein	4679	38
2018-08-28 11:00	ABUNGST-L.froth.1j	AlBungstein	4673	229
2018-08-28 11:00	ABUNGST-L.froth.1j	AlBungstein	4688	71
2018-08-28 11:00	ABUNGST-L.froth.1j	AlBungstein	4689	66
2018-08-28 11:00	ABUNGST-L.froth.1j	AlBungstein	5061	5
2018-08-28 11:00	ABUNGST-L.froth.1j	LOCAL SERVICE	4688	1
2018-08-28 11:00	ABUNGST-L.froth.1j	LOCAL SERVICE	4689	2
2018-08-28 11:00	ABUNGST-L.froth.1j	SYSTEM	4624	4
2018-08-28 11:00	ABUNGST-L.froth.1j	SYSTEM	4627	4

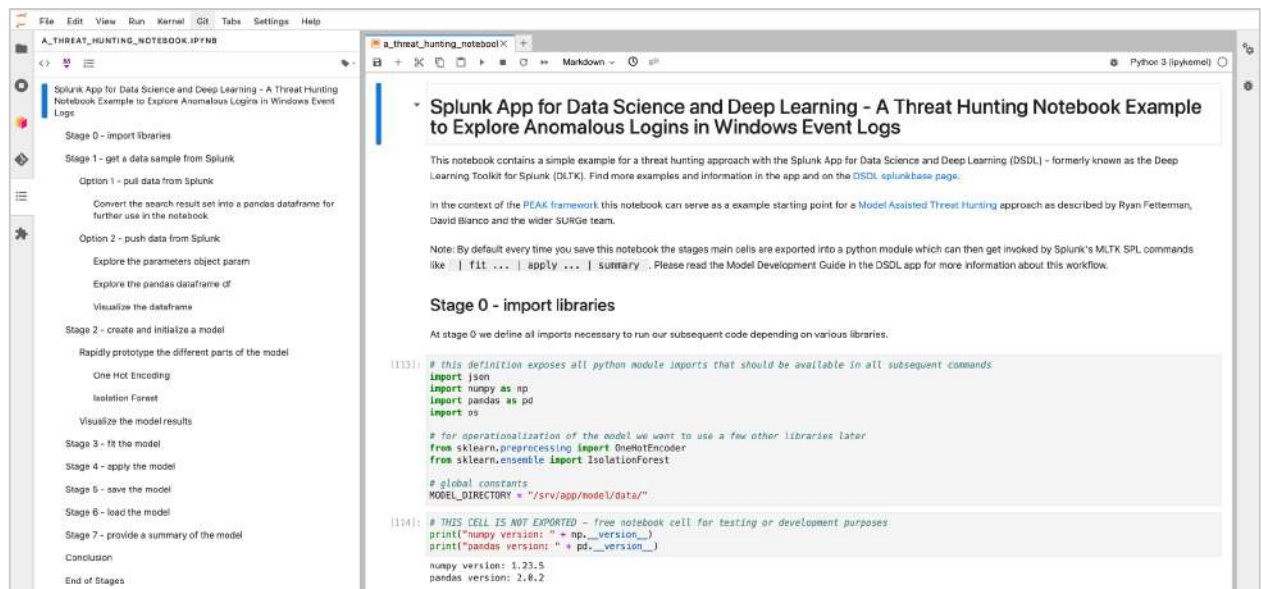
Step 3 - Prepare our environment with the right data

As a former data scientist you are familiar with Jupyter notebooks and want to use them for building your analysis and a model that can be improved with new data for long-term anomaly detection. In the DSDL app you can launch a container with Jupyter and other useful data science tools.

1. Check that your JupyterLab environment is available by navigating to the **Splunk App for Data Science and Deep Learning** and go to **Configuration > Containers**. You should see the `__dev__` container is up and running, and clicking on the `jupyter_url` in the table will launch the JupyterLab environment. Log on to this environment with the password “**Splunk4DeepLearning**”.



2. Once you access Jupyter Lab you find a notebook that a colleague has created for a similar threat hunting task, so you decide to use it for your goal. You open `a_threat_hunting_notebook.ipynb`.



3. We will need some data to use for this notebook, however, so navigate back to your Splunk environment and run the following search to send some data over to our development container so we can start threat hunting:

```
index=botsv3 source="*WinEventLog:Security"
| bin _time span=1h
| fillnull value=unknown Account_Name
| stats count by _time ComputerName Account_Name EventCode
| fit MLTKContainer mode=stage algo=a_threat_hunting_notebook _time ComputerName
Account_Name EventCode count into app:a_threat_hunt_model
```

Note that the `mode=stage` option means that we are not doing any further processing of the data in SPL, it indicates that we just want to send our results over to our development container. The data that is sent over can be found in the JupyterLab environment in the `/notebooks/data/` folder and you should see a file called `a_threat_hunting_model.csv`. Double click on this file and the data should be opened in a new tab. Quickly validate that it matches the results that are shown in Splunk before moving on.

Step 4 - Start exploring the data using the threat hunting notebook

At this point of the lab, you can follow along with the instructions in the Notebook to see how the sections and stages are working. Note that the “**Get data from Splunk**” section in the notebook has already been run above in Step 3!

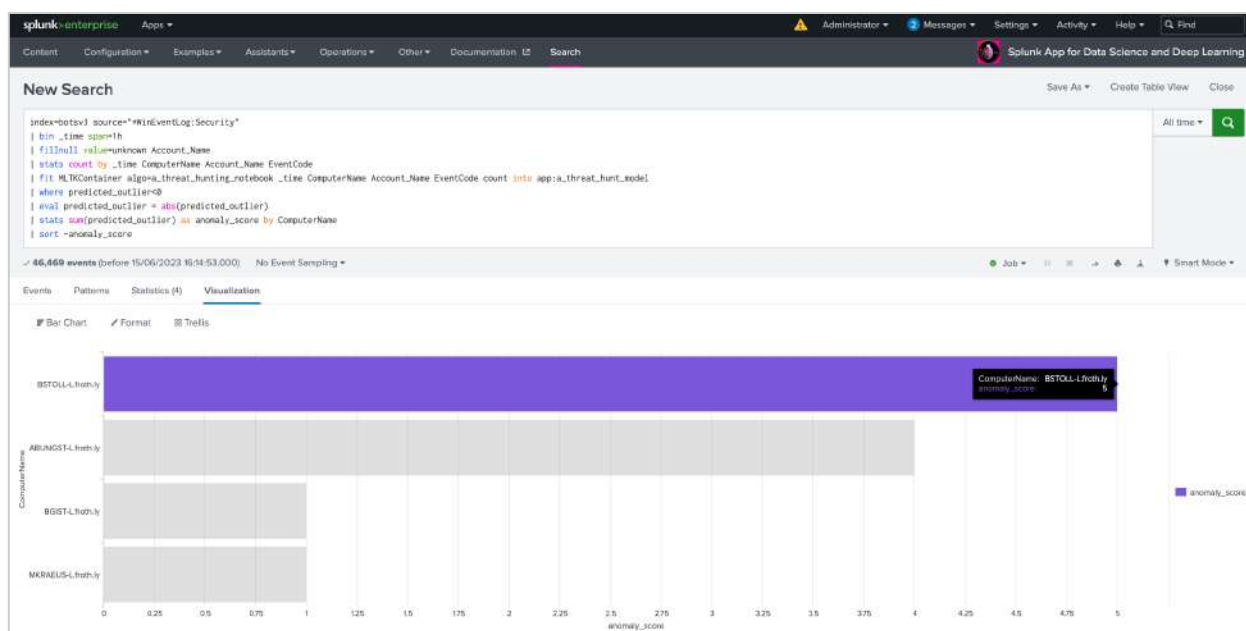
Move on to [step 5](#) of this lab when you’ve completed the notebook.

Step 5 - Conclusion

Congratulations! You have now seen how you can utilize the Splunk Platform and JupyterLab together to conduct your analysis and build a model to leverage and operationalize for your intended use case. For example, you can define a report or alert based on your model that flags Computers with the most anomalous logins with the following SPL command:

```
index=botsv3 source="*WinEventLog:Security"
| bin _time span=1h
| fillnull value=unknown Account_Name
| stats count by _time ComputerName Account_Name EventCode
| fit MLTKContainer mode=stage algo=a_threat_hunting_notebook _time ComputerName
Account_Name EventCode count into app:a_threat_hunt_model
| where predicted_outlier<0
| eval predicted_outlier = abs(predicted_outlier)
| stats sum(predicted_outlier) as anomaly_score by ComputerName
| sort -anomaly_score
```

This can be a critical starting point for further investigations, triggering a notable event in Enterprise Security, or adding a risk score in case you use risk based alerting (RBA) in Enterprise Security.



(Optional) Use Case 4: Analyzing Network Traffic

Scenario

Working in the network monitoring team at Buttercup Bank you have been asked to see if it is possible to identify unusual network behaviour after the Network Operations Centre (NOC) received multiple complaints about network performance. With a background in data science you would prefer to run your analysis using Python rather than SPL, but appreciate that the rest of the team uses Splunk for their day-to-day monitoring.

You decide to use the Splunk App for Data Science and Deep Learning to run your analysis, given it provides you with a JupyterLab environment, but if your work results in a machine learning model you will save it in an ONNX format so that it can be deployed back in the Splunk platform via MLTK.

Steps

1. Check that your JupyterLab environment is available by navigating to the **Splunk App for Data Science and Deep Learning** and go to **Configuration > Containers**. You should see the **__dev__** container is up and running, and clicking on the **jupyter_url** in the table will launch the JupyterLab environment. Log on to this environment with the password “**Splunk4DeepLearning**”.
2. Once you access Jupyter Lab you find a notebook that you worked on previously called **a_network_analysis.ipynb**. Open this notebook and follow through the steps within to train an **sklearn** pipeline for detecting unusual network activity, save it as an ONNX model file that can be uploaded to Splunk and then run some additional analysis using **PCA**, **t-SNE** and **UMAP** to confirm that your outlier detection model is working as expected by showing separable outliers in the network traffic data.