Tutorial on Machine Learning Tools

Yanbing Xue

Milos Hauskrecht

Why do we need these tools?

- Widely deployed classical models
- No need to code from scratch
- Easy-to-use GUI

Outline

- Matlab Apps
- Weka 3 UI
- TensorFlow

Outline

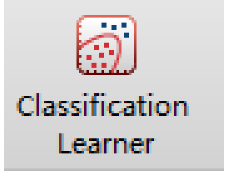
- Matlab Apps
  - Introduction
  - App for Classification (Classification Learner)
  - App for Regression (Regression Learner)
  - App for Time Series (Neural Net Time Series)
- Weka 3 UI
- TensorFlow

Matlab Apps - Introduction

- What are Matlab apps?
  - A set of apps that can complete basic tasks related to machine learning
  - For machine learning models:
    - Classification
    - Regression
    - Clustering
    - Time series
  - For Applications:
    - Signal processing, biomedical computing, marketing, etc

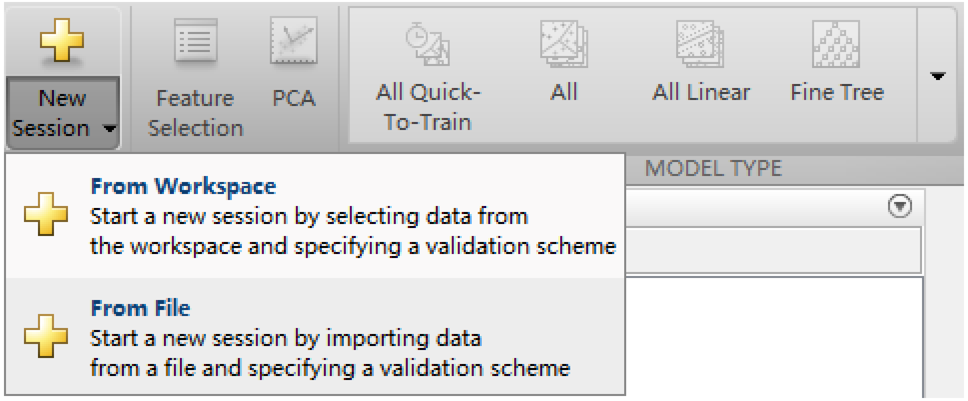
Classification Learner

- An app including multiple classical classifiers:
  - Logistic regression, decision tree, discriminant analysis, SVM, kNN, ensembling methods, etc
- We can launch classification learner by clicking



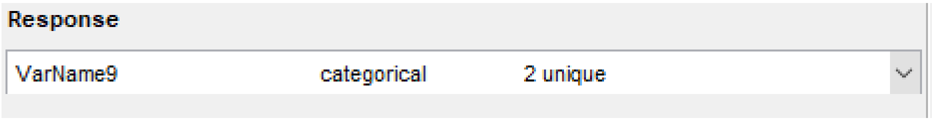
Workflow

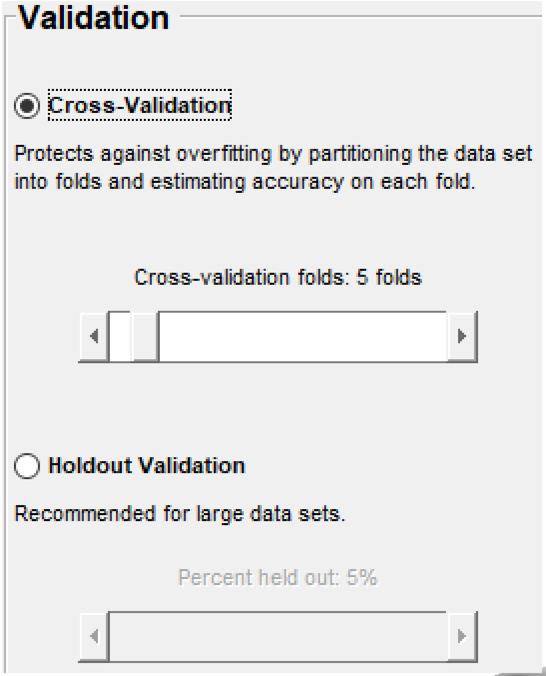
- Step 1: importing data
  - From a matrix in current workspace, OR
  - From an external file (.xls, .txt, .csv, etc)



Workflow

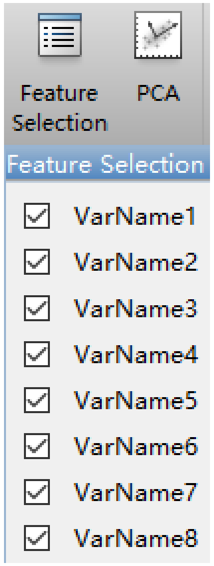
- Step 2:
  - Specify the target (response)
  - Specify the validation
    (K-fold cross validation or x% holdout validation)

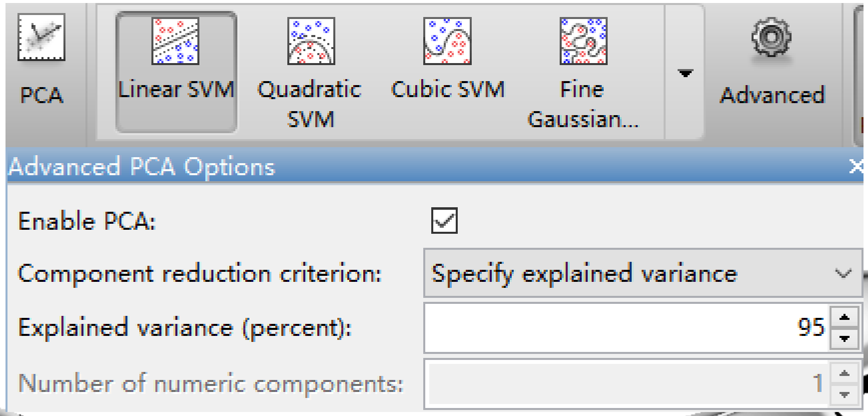




Workflow

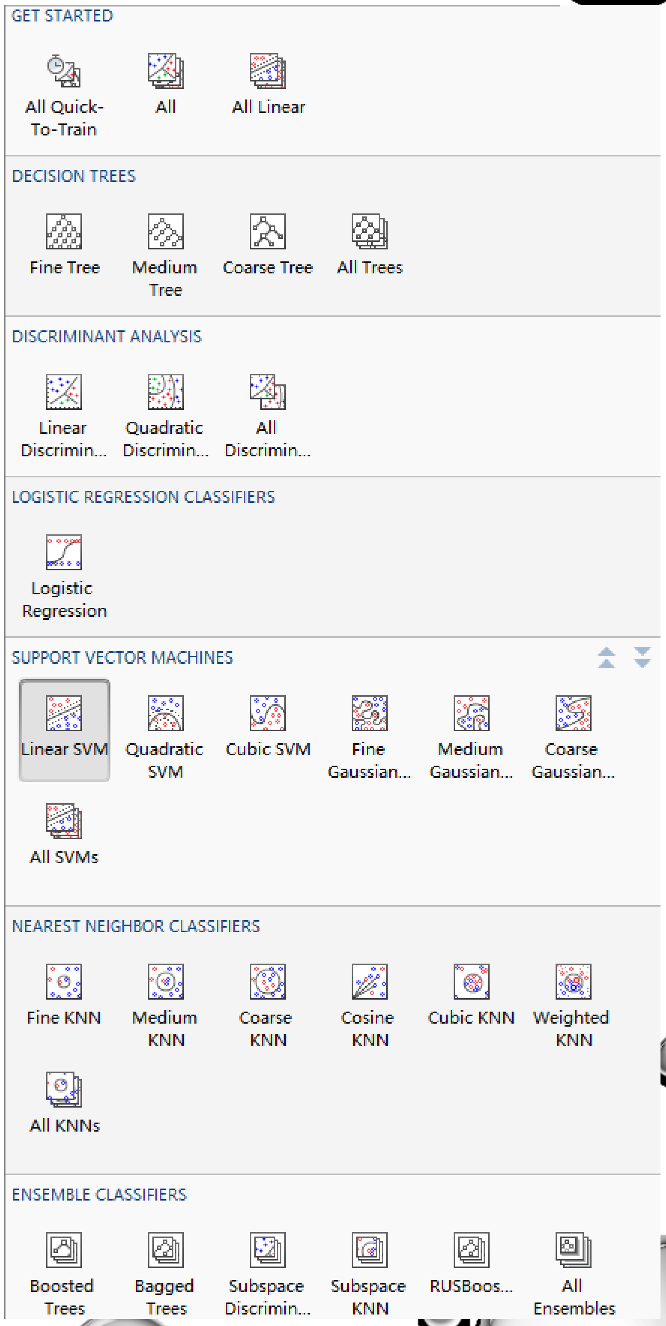
- Step 3: feature reduction
  - Feature selection
    - Directly remove certain features
  - PCA
    - Specify the proportion of information preserved (explained variance), OR
    - Specify the number of components preserved





Workflow

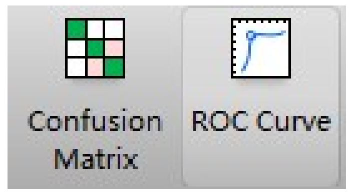
- Step 4: model selection



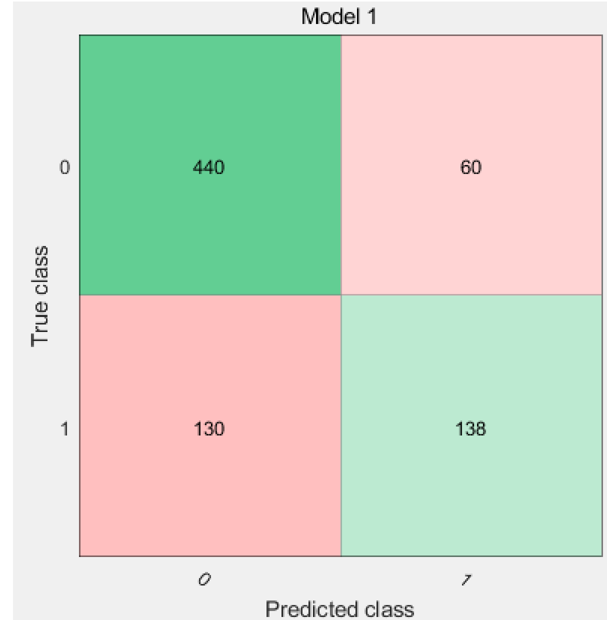
- Step 5: start training
  - By clicking

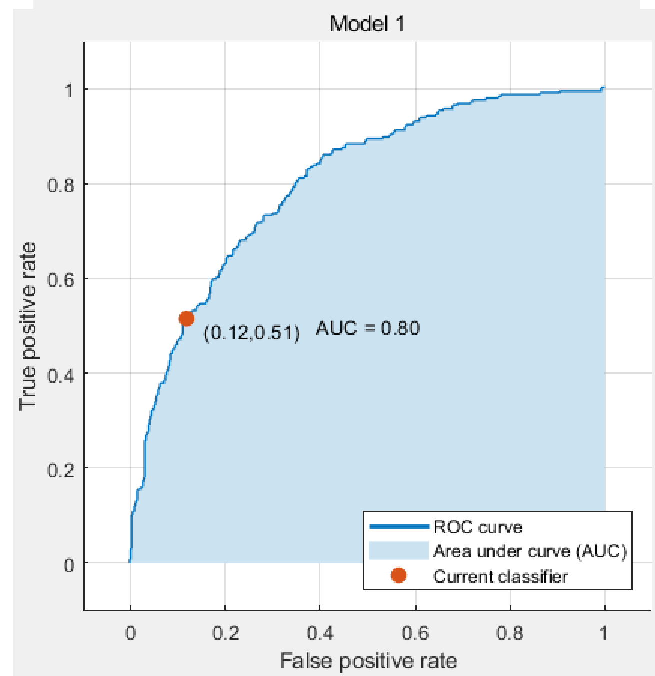
Workflow

- Step 6: plotting



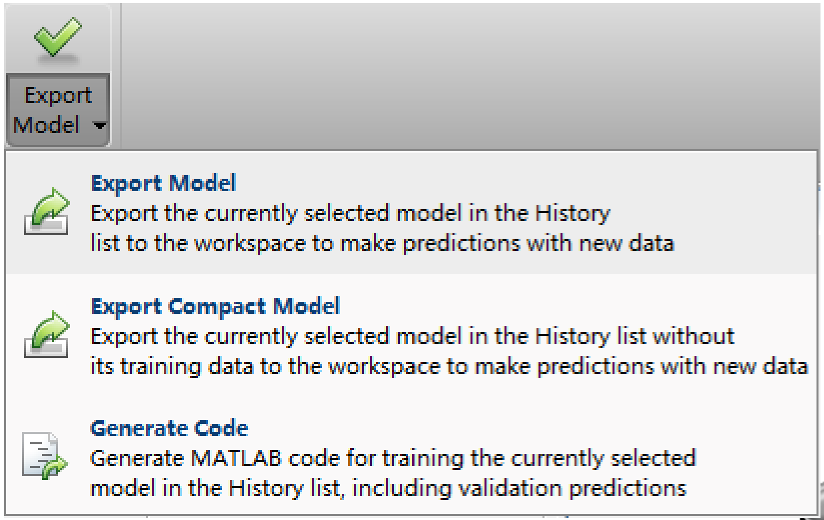
  - Confusion matrix
  - ROC Curve





Workflow

- Step 7: export model
  - To workspace, OR
  - To workspace w/o training data, OR
  - Generate code

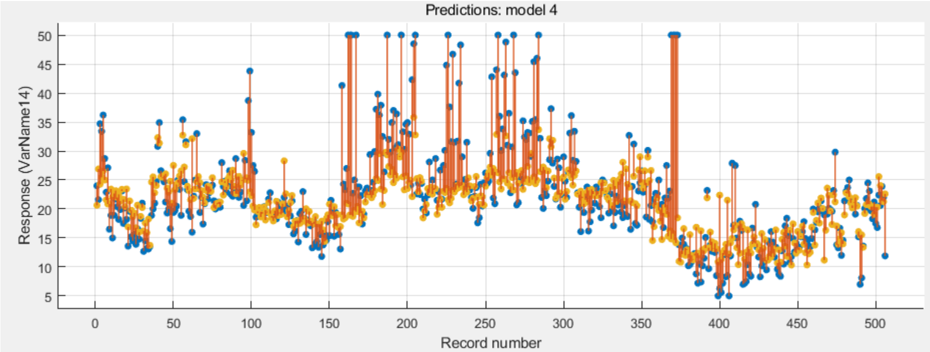


Regression Learner

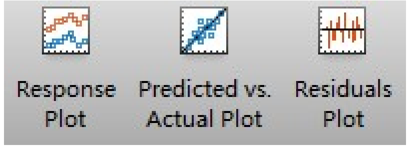
- An app including multiple classical regression models:
  - Linear regression, SVM, Gaussian process, regression tree, ensembling trees, etc

- We can launch regression learner by clicking

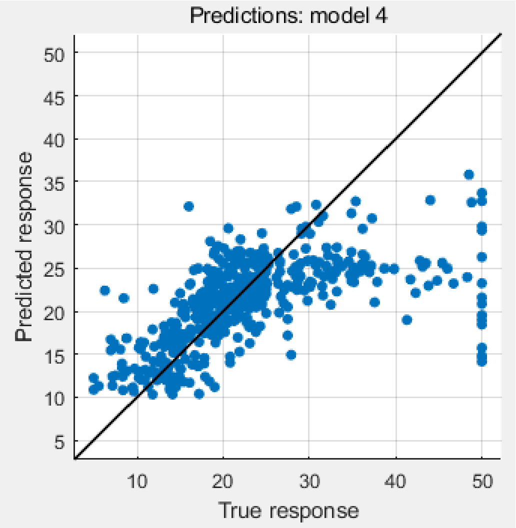
Workflow

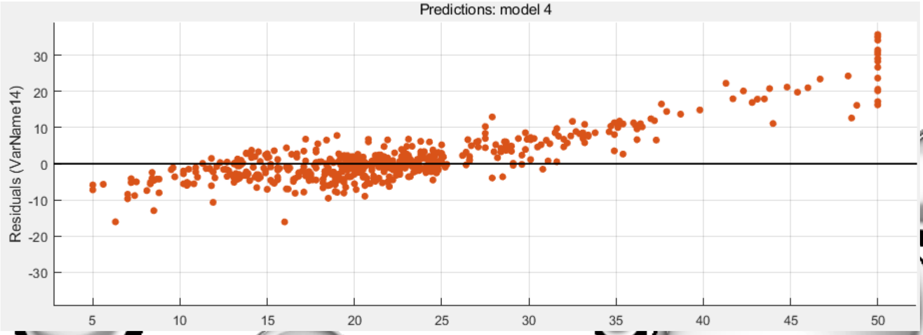


- Step 1 ~ 5 are the same
- Step 6: plotting



  - Response plot
    - True & predicted for each instance
  - Predicted vs actual plot
    - Predicted for each true value
  - Residuals plot
    - (True – predicted) for each true value



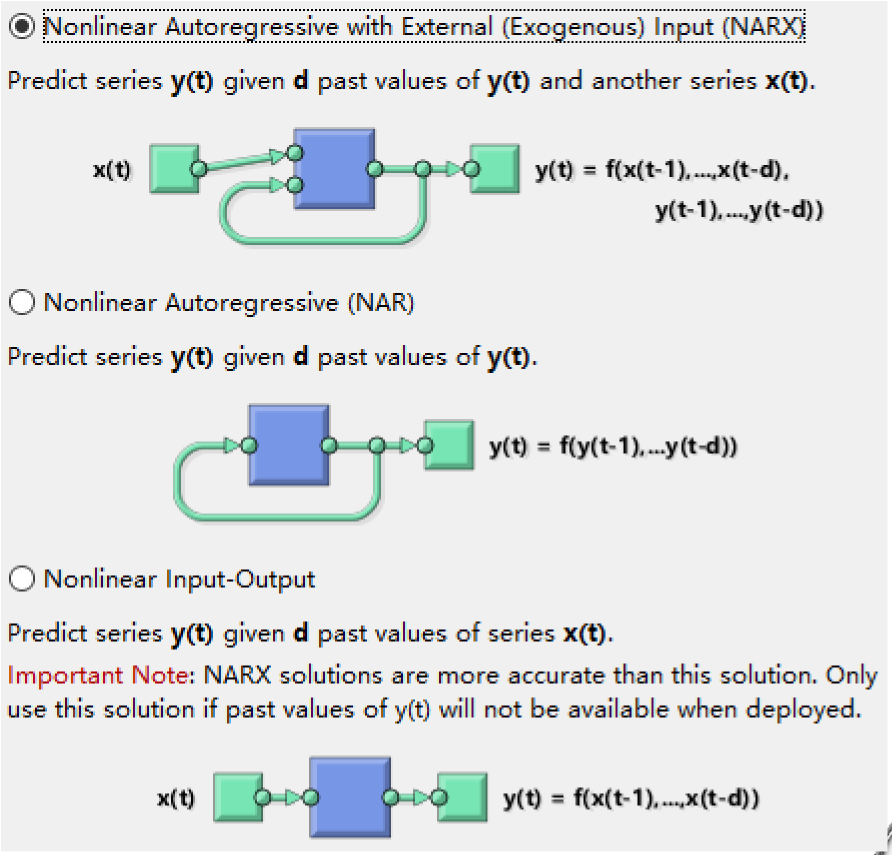


Neural Net Time Series

- What is time series?
- In time series, data instances are time dependent
  - In classification & regression
    - Output of current instance <= input of current instance
  - In time series
    - Output of current instance <= input of current instance & output of previous instances & input of previous instances
- We can launch neural net time series by clicking

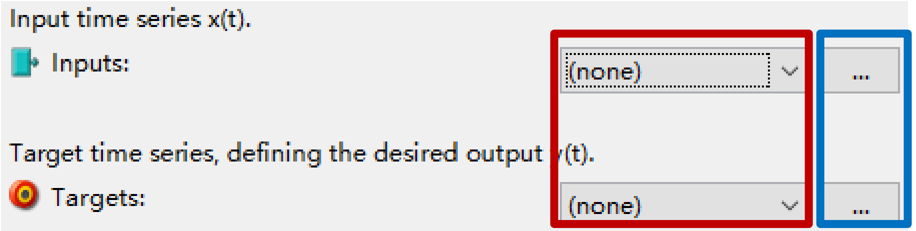
Workflow

- Step 1: model selection
  - NARX (recommended)
    - Both input and output dependent
  - NAR
    - Only output dependent
  - NIO
    - Only input dependent



Workflow

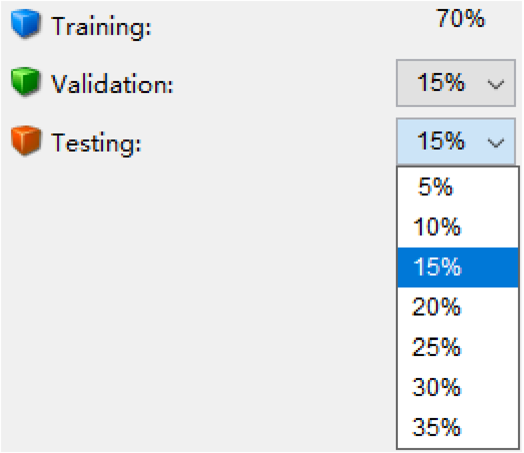
- Step 2: data loading
  - From workspace, OR
  - From file



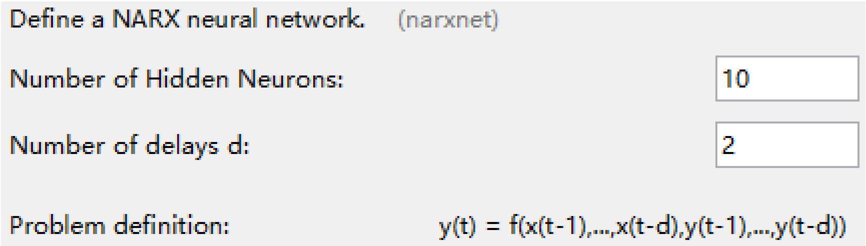
- Step 3: set feature format
  - Cell (scalar)
  - Column vector
  - Row vector

Workflow

- Step 4: set validation

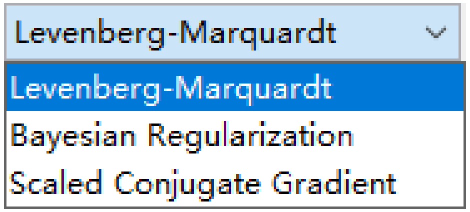


- Step 5: set architecture
  - # of hidden units
  - # of delay
    - # of dependent previous instances



Workflow

- Step 6: select algorithm



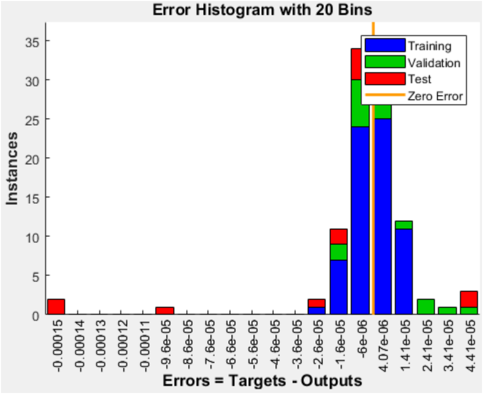
- Step 7: start training
  - By clicking

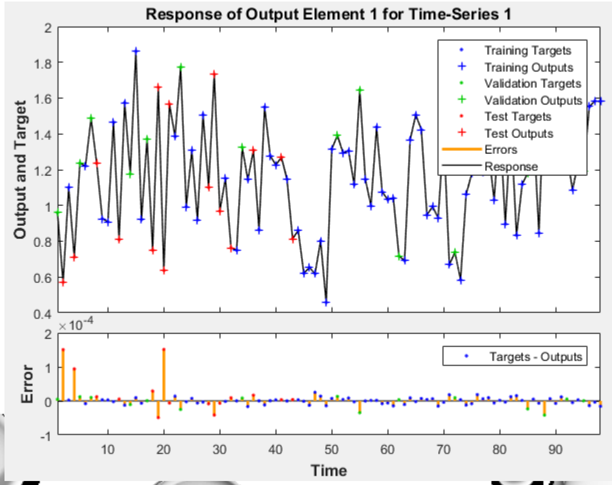
Workflow

	Target Values	MSE	R
Training:	70	8.10310e-11	9.99999e-1
Validation:	15	3.25189e-10	9.99999e-1
Testing:	15	4.00371e-9	9.99999e-1

Plot Error Histogram | Plot Response

- Step 8: result visualization
  - Error histogram
    - Distribution of errors
  - Response
    - True and predicted for each instance





- Step 9: generate function

Outline

- Matlab Apps
- Weka 3 UI
  - Introduction
  - Weka Explorer
  - Weka KnowledgeFlow
- TensorFlow

Weka - Introduction

- What is Weka?
  - A set of Java APIs for machine learning and data mining with several GUIs
  - For machine learning models:
    - Classification
    - Regression
    - Clustering
    - Rule-based models

Weka Explorer

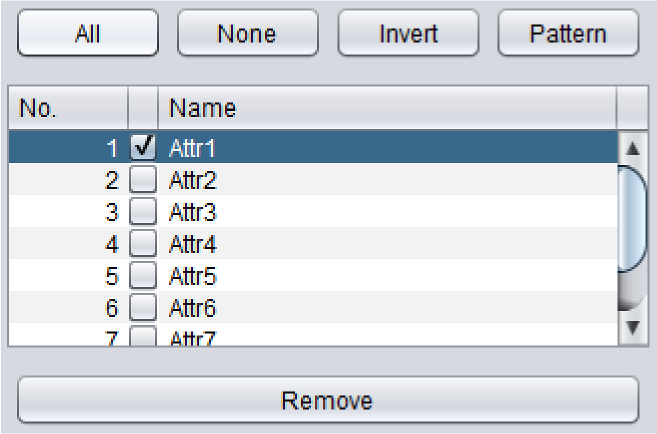
- A more user-friendly GUI
  - Multiple data source (local, URL, JDBC, etc)
  - Complex data preprocessing techniques
  - Machine learning models (SVM, regression, tree, instance-based, rule-based, Bayesian, etc)

- We can launch explorer by clicking

Workflow

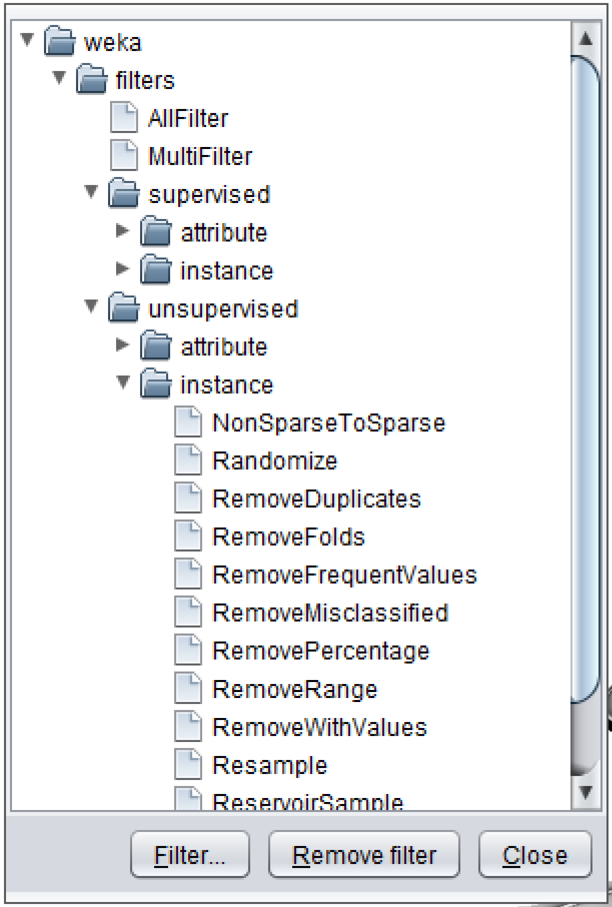
- Step 1: data loading

- Step 2: feature selection
  - Remove selected features



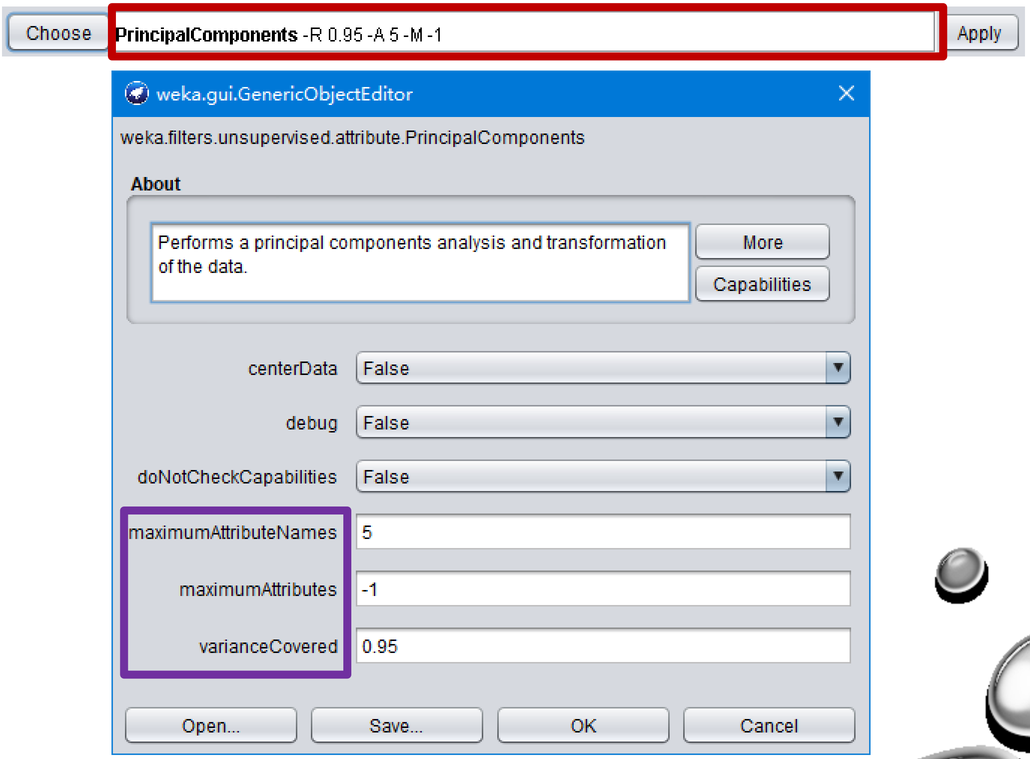
Workflow

- Step 3: filtering
  - Conversion (e.g. numeric to nominal)
    - Essential to classification
  - Normalization
  - Discretization
  - PCA
  - …



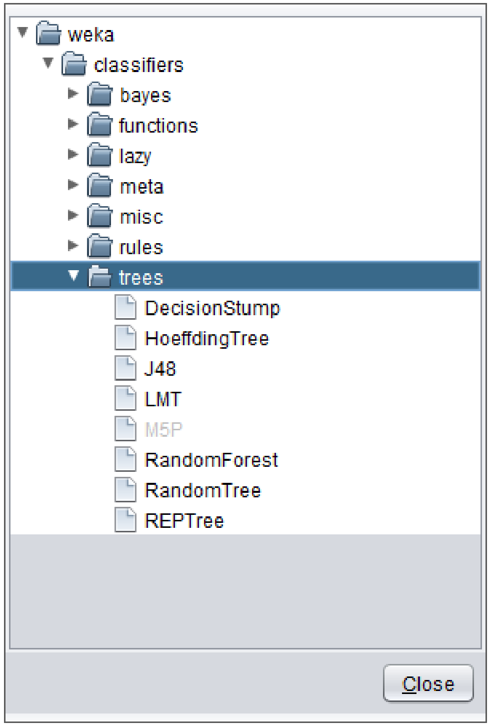
Workflow

- Click for advanced settings
- Hover over the name for description
- You can use multiple filters consequently



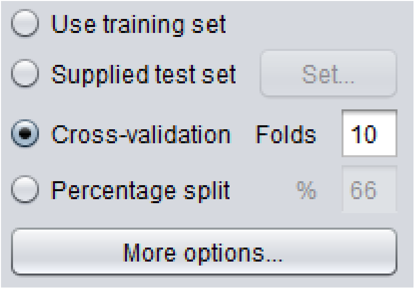
Workflow

- Step 4: model selection
  - Gray items are unavailable
  - Click for advanced settings

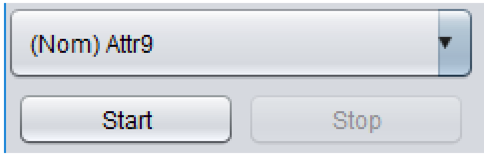


Workflow

- Step 5: set validation
  - K-fold or x% split

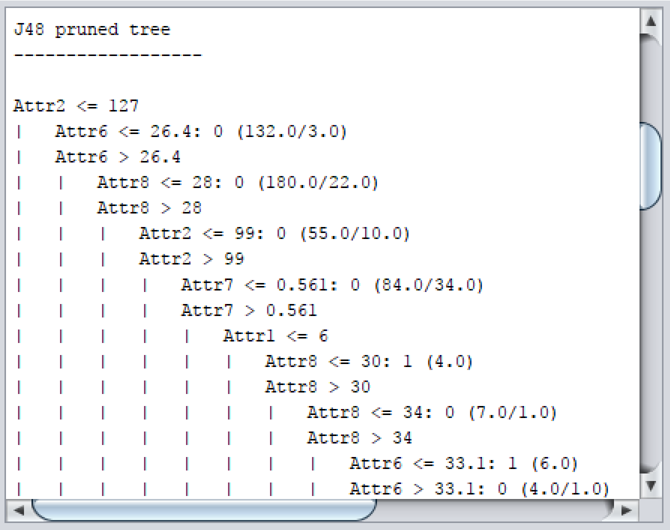


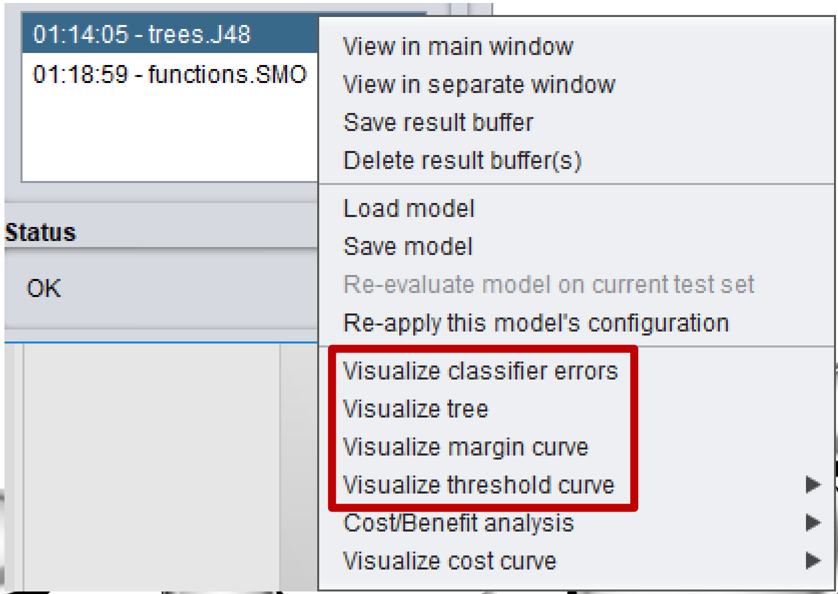
- Step 6: select target



Workflow

- Step 7: output
  - Tree structure (if any)
  - Error rate
  - Confusion matrix
- Step 8: visualization
  - True and predicted in 2-d feature space
  - Tree structure (if any)
  - Marginal distribution
  - ROC





Weka KnowledgeFlow

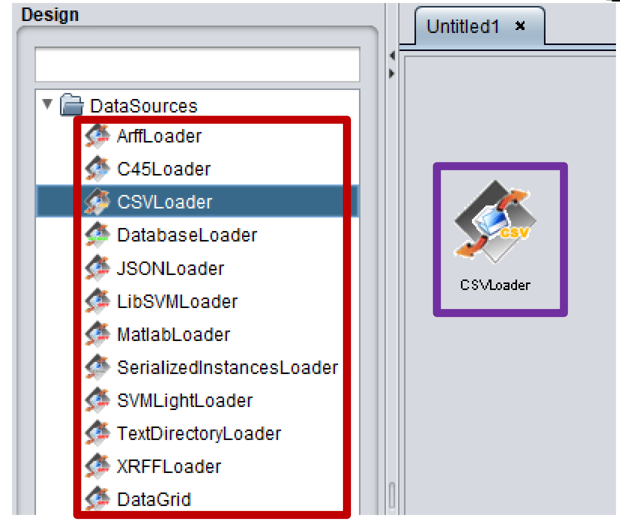
- Disadvantage of Weka explorer
  - Must load the whole dataset into memory
  - Cannot control the workflow among different models

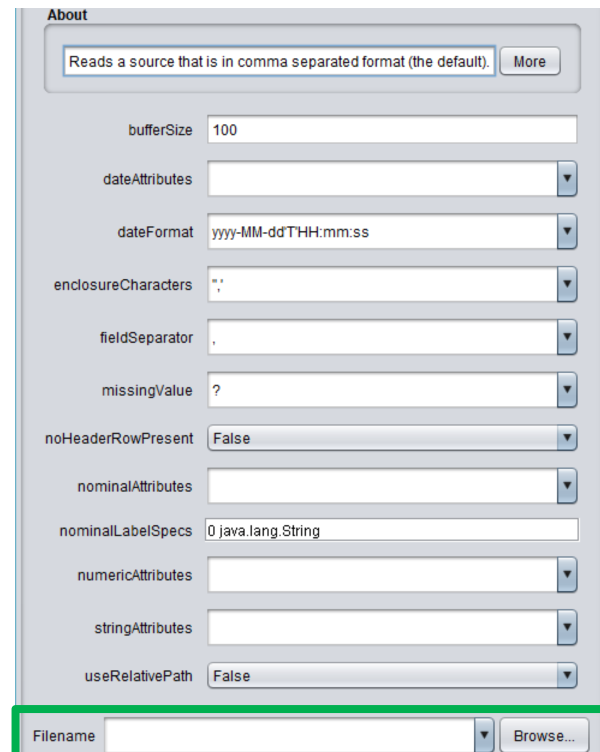
- Weka KnowledgeFlow is a GUI for building medium-large projects on large datasets

- We can launch knowledgeFlow by clicking

Workflow

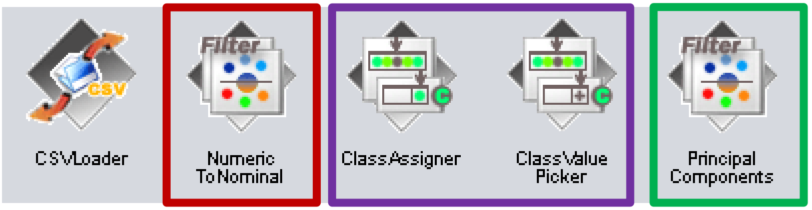
- Step 1: data loading
  - Pick a loader for your data
  - Click to draw it on the panel
  - Double click for advanced settings
  - Select the data file





Workflow

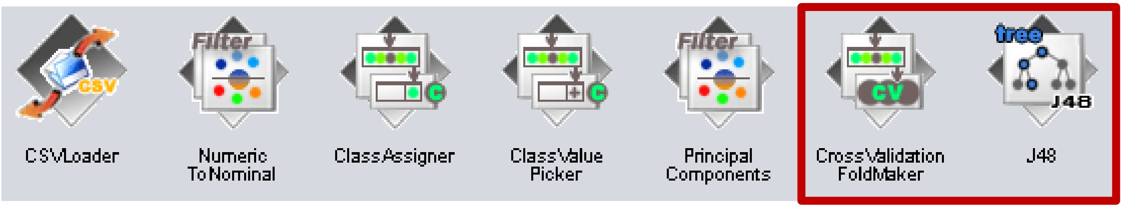
- Step 2: target settings
  - In “Filters”
  - Convert target to categorical
  - In “Evaluation”
  - Set target attribute
  - Set positive class



- Step 3: add other filters

Workflow

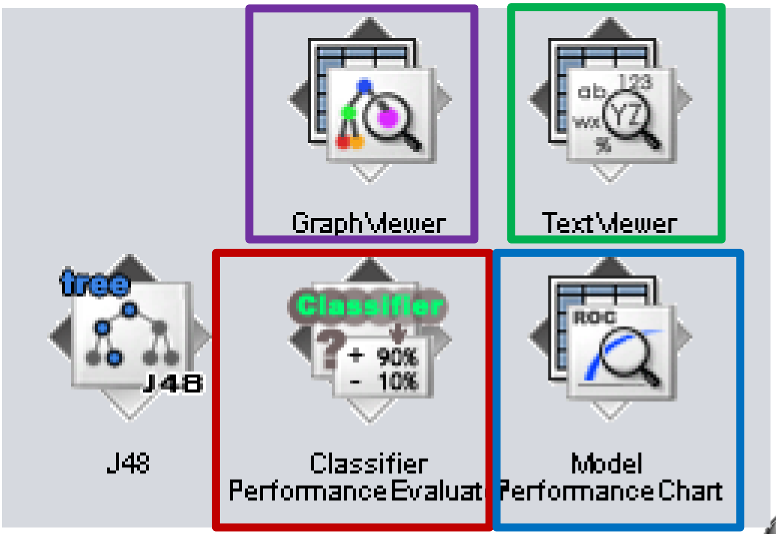
- Step 4: validation settings
  - In “Evaluation”



- Step 5: model selection
  - You must manually check the availability of the model

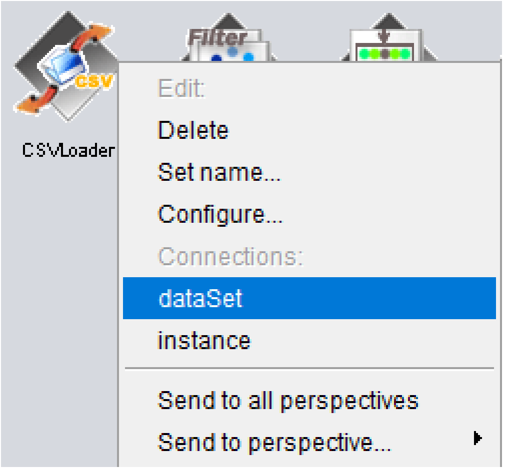
Workflow

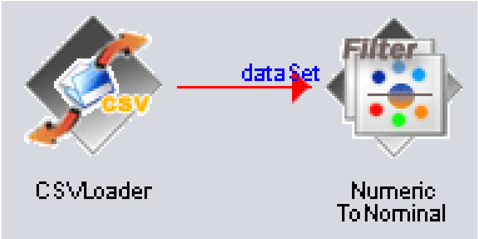
- Step 6: evaluation
    - In "Evaluation"
    - Generate error rate, confusion matrix, ROC, etc
- Step 7: visualization
    - In "Visualization"
        - Model structure
        - Statistics in texts
        - ROC and other plots



Workflow

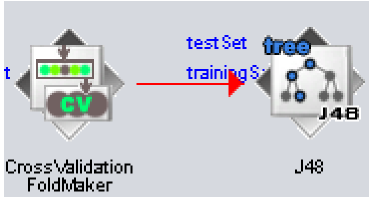
- Step 8: connections
  - Right click each module and select an output type
  - For data source:
    - Load the whole data, OR
    - Load by instance

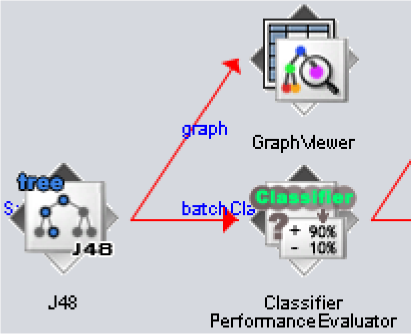


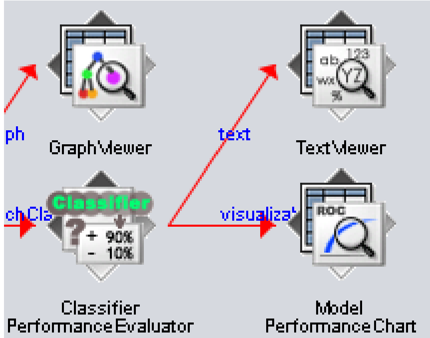


Workflow

- For cross validation:
  - Output both training and testing set to the model
- For model:
  - Output graph for visualizing the structure
  - Output batchClassifier for evaluation
- For evaluation
  - Output text for error rates and confusion matrix
  - Output visualizableError for ROC and other plots

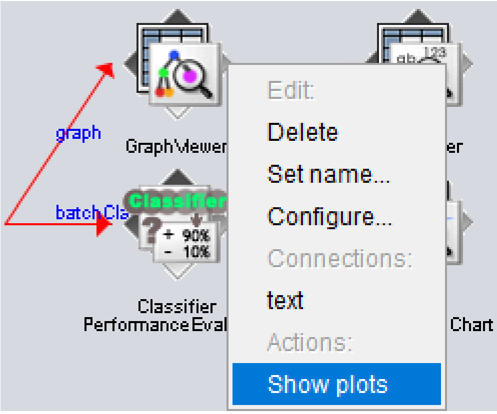


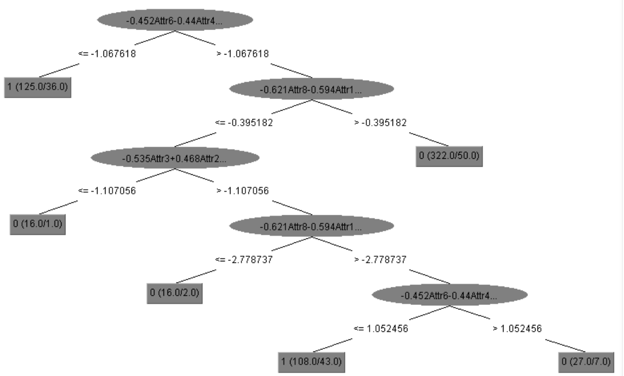




Workflow

- Step 9: model building
  - Run the flow
  - Right click all the three viewers for results





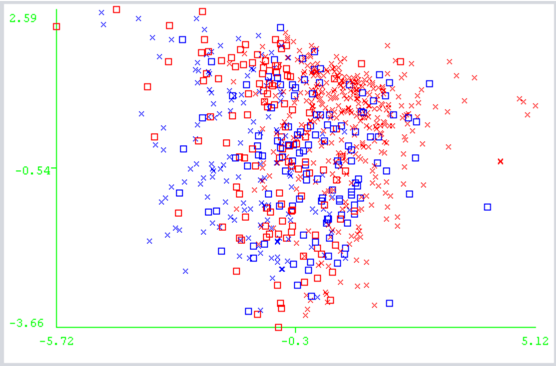

```
Incorrectly Classified Instances       243               31.6406 %
Kappa statistic                          0.3006
Mean absolute error                      0.3788
Root mean squared error                  0.4632
Relative absolute error                 83.3436 %
Root relative squared error             97.188  %
Total Number of Instances              768

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC
                 0.537    0.238    0.548      0.537    0.542      0.301
                 0.762    0.463    0.754      0.762    0.758      0.301
Weighted Avg.    0.684    0.384    0.682      0.684    0.683      0.301

=== Confusion Matrix ===

   a   b   <-- classified as
 144 124 |   a = 1
 119 381 |   b = 0
```



Outline

- Matlab Apps
- Weka 3 UI
- TensorFlow
  - Introduction
  - Example – Linear Regression
  - Example - CNN

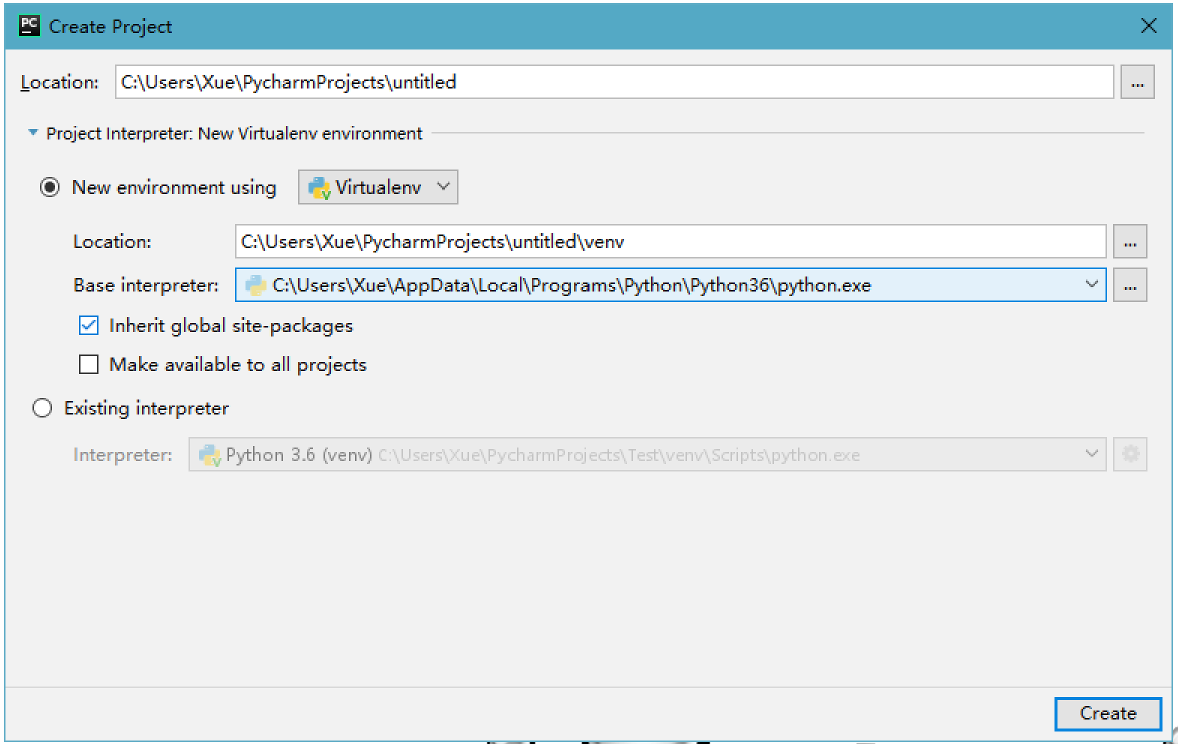
TensorFlow - Introduction

- A multi-language (mainly for Python 3 64bit) API for the newest deep learning algorithms
  - Convolutional Neural Network
  - Recurrent Neural Network
  - Long-Short Term Memory
  - Autoencoder
  - …

* For details of functions, please refer to TensorFlow API and NumPy API. We do not have enough time explaining each line of the codes in this tutorial

Use TensorFlow in PyCharm

- File – New Project
  - Select "Inherit global site-packages"



Example – Linear Regression

- Generate 100 random scalars as x, and y=0.1x+0.3

```
import tensorflow as tf
import numpy as np

x_data = np.random.rand(100).astype(np.float32)
y_data = x_data*0.1 + 0.3
```

- Set w (starting from random) and b (starting from 0)

```
Weights = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
biases = tf.Variable(tf.zeros([1]))

y = Weights*x_data + biases
```

- Set mean squard error

```
loss = tf.reduce_mean(tf.square(y-y_data))
```

- Set gradient descent (with step length of 0.5)

```
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)
```

Example – Linear Regression

- Set initializer

```
init = tf.global_variables_initializer()
```

- Start the workflow

```
sess = tf.Session()
sess.run(init)
```

- Run for 201 steps and print per 20 steps

```
for step in range(201):
    sess.run(train)
    if step % 20 == 0:
        print(step, sess.run(Weights), sess.run(biases))
```

- Output:

```
0 [ 0.62302148] [ 0.06378302]
20 [ 0.23029616] [ 0.23874463]
40 [ 0.13269234] [ 0.28463054]
60 [ 0.10820277] [ 0.29614368]
80 [ 0.10205815] [ 0.29903242]
100 [ 0.1005164] [ 0.29975724]
120 [ 0.10012957] [ 0.2999391]
140 [ 0.10003252] [ 0.29998472]
160 [ 0.10000816] [ 0.29999617]
180 [ 0.10000206] [ 0.29999906]
200 [ 0.10000052] [ 0.29999977]
```

Example - CNN

- Recall CNN in previous slides
  - Convolution layer
  - Activation function
  - Pooling layer
  - Fully connected layer

- In this example, we use MNIST data
  - 784 features (28*28*1 gray-scale image)
  - 10 classes (digit 0 ~ digit 9)

CNN - Preprocessing

- Import MNIST data

```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

- Set dimension # of x and class # of y

```
xs = tf.placeholder(tf.float32, [None, 784])
ys = tf.placeholder(tf.float32, [None, 10])
```

- Reshape x into 28*28*1

```
x_image = tf.reshape(xs, [-1, 28, 28, 1])
```

CNN – Convolution Layer

- Set weight & bias
  - Size 5*5*1, 32 kernels

```
W_conv1 = tf.Variable(tf.truncated_normal([5, 5, 1, 32], 0, 0.1))
b_conv1 = tf.Variable(tf.zeros(32))
```

- Set activation function
  - Rectified Linear Unit (ReLU)
  - 28*28*32 features

```
h_conv1 = tf.nn.relu(tf.nn.conv2d(x_image, W_conv1, [1, 1, 1, 1], 'SAME') + b_conv1)
```

- Set pooling
  - 2*2 => 1*1
  - 14*14*32 features

```
h_pool1 = tf.nn.max_pool(h_conv1, [1, 2, 2, 1], [1, 2, 2, 1], 'SAME')
```

CNN – Fully Connected Layer

- Flatten all the convolution features

```
h_pool1_flat = tf.reshape(h_pool1, [-1, 14*14*32])
```

- Set weight & bias & activation function
  - 14*14*32 => 1024 featur
  - No convolution

```
W_fc1 = tf.Variable(tf.truncated_normal([14*14*32, 1024], 0, 0.1))
b_fc1 = tf.Variable(tf.zeros(1024))
h_fc1 = tf.nn.relu(tf.matmul(h_pool1_flat, W_fc1) + b_fc1)
```

CNN – Output Layer

- Set weight & bias
  - 10 outputs

```
W_fc2 = tf.Variable(tf.truncated_normal([1024, 10], 0, 0.1))
b_fc2 = tf.Variable(tf.zeros(10))
```

- Set predictions
  - Softmax function

```
prediction = tf.nn.softmax(tf.matmul(h_fc1, W_fc2) + b_fc2)
```

CNN – Model Settings

- Set generalization error
  - Cross entropy

```
cross_entropy = tf.reduce_mean(-tf.reduce_sum(ys*tf.log(prediction), [1]))
```

- Set training algorithm
  - Adam (step length 1e-4)

```
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
```

- Run for 501 steps and report accuracy per 50 steps

```
sess = tf.Session()
sess.run(tf.global_variables_initializer())
for i in range(501):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={xs: batch_xs, ys: batch_ys})
    if i % 50 == 0:
        y_pre = sess.run(prediction, feed_dict={xs: mnist.test.images})
        res = tf.equal(tf.argmax(y_pre, 1), tf.argmax(mnist.test.labels, 1))
        accuracy = tf.reduce_mean(tf.cast(res, tf.float32))
        print(sess.run(accuracy, feed_dict={xs: mnist.test.images}))
```

Questions?