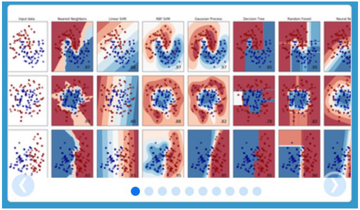
Machine Learning With Scikit-Learn and TensorFlow

Anurag Nagar

Machine Learning Class

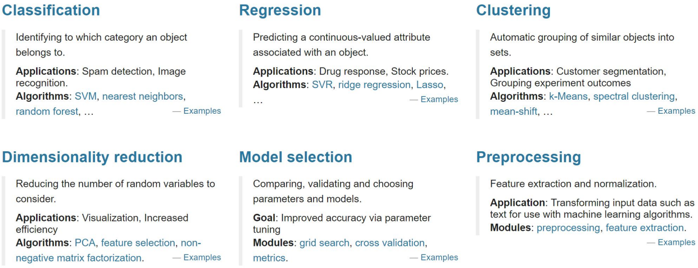
Introduction

- One of the most popular ML libraries is scikit-learn
- Python based
- Built on familiar Python libraries, such as NumPy, SciPy, and matplotlib



Libraries

- Libraries for most ML tasks
- Efficient algorithms and parameter tuning
- Can be used as part of a workflow with pipelines



Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.
Algorithms: SVM, nearest neighbors, random forest, ...

— Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, ridge regression, Lasso, ...

— Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, ...

— Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: PCA, feature selection, non-negative matrix factorization.

— Examples

Model selection

Comparing, validating and choosing parameters and models.

Goal: Improved accuracy via parameter tuning
Modules: grid search, cross validation, metrics.

— Examples

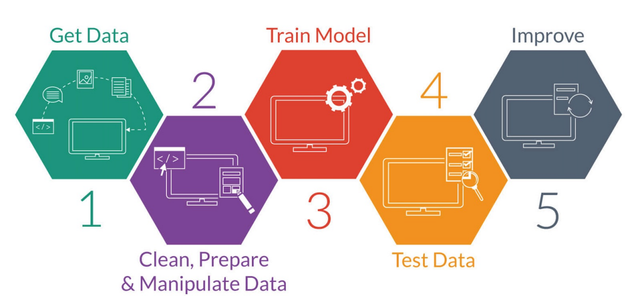
Preprocessing

Feature extraction and normalization.

Application: Transforming input data such as text for use with machine learning algorithms.
Modules: preprocessing, feature extraction.

— Examples

Steps in ML Project



- Libraries for most ML tasks
- Efficient algorithms and parameter tuning
- Can be used as part of a workflow with pipelines

Getting the data

Library with easy-to-use data structures and data analysis tools.

```
import pandas as pd
```

- Great way to read data files into a **dataframe**.
- One of the frequently used methods:

  `df = pd.read_csv(filepath, other_parameters)`

  other_parameters can specify delimiters, header, index_col, quotechar, etc

  More details can be found in the documentation

DataFrames

DataFrames are powerful data structures in Python, permit extraction and manipulation of structured and semi-structured data. **Some examples:**

- Extracting a column by name:
  `df['column_name']`
- Extracting a column by index:
  `df.iloc[,1:1]`
- Describe the details of each feature
  `df.describe()`

Built-in Datasets

Scikit-Learn comes with a handy set of built in datasets

```
from sklearn import datasets
```

- Great way to read data files into a **dataframe**.
- One of the frequently used methods:

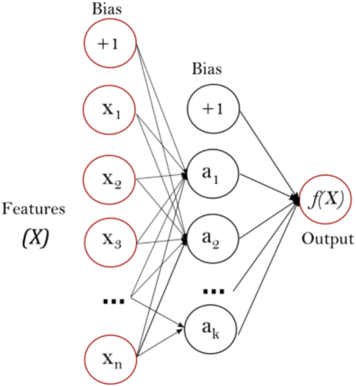
  ```
  df = pd.read_csv(filepath, other_parameters)
  ```

  other_parameters can specify delimiters, header, index_col, quotechar, etc

  More details can be found in the documentation

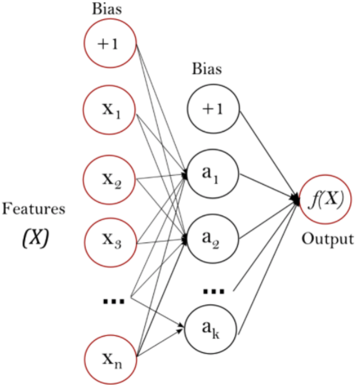
Artificial Neural Network

- ANN is a supervised learning algorithm that tries to learn a function $f : R^m \rightarrow R^o$ where m is the number of input features and o is the number of outputs.
- Non-linear function approximator.
- Network consists of input, hidden and output Layers.



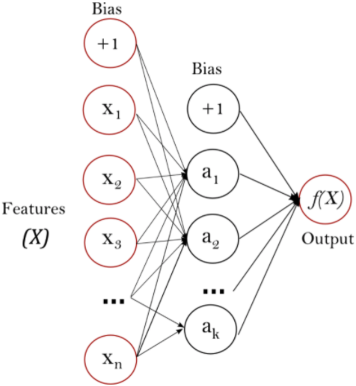
Artificial Neural Network

- Each neuron in the hidden layer transforms the values from previous layer as the weighted sum $\sum_{\imath=0}^{n} w_i x_i$, followed by a non-linear activation function g. Note: w_0 is the bias term



Artificial Neural Network

- Advantages :
  - Ability to learn complex non-linear datasets
  - Highly adaptable
- Disadvantages :
  - Loss functions are non-convex, could lead to local minima
  - A larger number of hyperparameters require tuning
  - Feature scaling is often required



Multi-layer Perceptron

- In SciKit Learn, ANN is referred to as Multi-layer Perceptron (MLP).
- Details about the class are available here.
- Some of the key parameters are:
  hidden_layer_sizes : tuple indicating size of each hidden layer
  activation: activation function to be used
  solver: type of solver to be used

```
from sklearn.neural_network import
    MLPClassifier
X = [[0., 0.], [1., 1.]]
y = [0, 1]
clf = MLPClassifier(solver='lbfgs', alpha=1e
    -5, hidden_layer_sizes=(5, 2), random_state
    =1)
```

More Complex Example

- A more interesting example.

```
from sklearn.datasets import
    load_breast_cancer
cancer = load_breast_cancer()
# let's look at the keys
cancer.keys()
# let's get the data and see dimensions
cancer['data'].shape
# We can also import the data into a dataframe
import numpy as np
import pandas as pd
df = pd.DataFrame(cancer.data, columns=cancer.
    feature_names)
# use describe on dataframe
df.describe()
# let's get just the data and target labels
X = cancer['data']
```

More Complex Example

- Let's apply the MLP algorithm on the data
- Note that scaling of data is essential when using MLP

```
# split data into train and test
from sklearn.model_selection import
    train_test_split
X_train, X_test, y_train, y_test =
    train_test_split(X, y, test_size = 0.2)
# scale data
from sklearn.preprocessing import
    StandardScaler
scaler = StandardScaler()
# Fit only to the training data
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
# time to train the classifier
```

Model Evaluation

- Let's analyze the results
- There's a really nice package called **sklearn.metrics** that can be used to evaluate your model.
- See more details here.

```
from sklearn.metrics import
    classification_report, confusion_matrix
print(confusion_matrix(y_test, predictions))
# full report
print(classification_report(y_test, predictions
    ))
```

TensorFlow

TensorFlow is a machine learning library using data flow graphs

Nodes represent numerical operations and edges represent the data arrays communicated between them.

Flexible Architecture allows for easy deployment to one or more CPUs or GPUs. Very high computational power and efficiency can be achieved.

Main unit of data flow is the

Tensor's Rank

- A tensor consists of a set of primitive values shaped into an array of any number of dimensions.
- A tensor's **rank** is its number of dimensions

```
3 # a rank 0 tensor; this is a scalar with shape []
[1., 2., 3.] # a rank 1 tensor; this is a vector with shape [3]
[[1., 2., 3.], [4., 5., 6.]] # a rank 2 tensor; a matrix with shape [2, 3]
[[[1., 2., 3.]], [[7., 8., 9.]]] # a rank 3 tensor with shape [2, 1, 3]
```

Tensor Flow Operations

Creating a TensorFlow application consists of two steps:

- Building the computational graph.
- Running the computational graph.

```
node1 = tf.constant(3.0, dtype=tf.float32)
node2 = tf.constant(4.0)
# start a session
sess = tf.Session()
print(sess.run([node1, node2]))
node3 = tf.add(node1, node2)
print("node3:", node3)
print("sess.run(node3):", sess.run(node3))
```

TensorFlow for Machine Learning

- TensorFlow comes with a set of **optimizers** e.g. **gradient descent** that can be used to train algorithms
- **tf.estimator** is a high-level TensorFlow library that simplifies the mechanics of machine learning, such as running training and testing loops, and managing data sets.
- Below is the listing that sets up an estimator for linear regression.

```
import tensorflow as tf
# NumPy is often used to load, manipulate and
    preprocess data.
import numpy as np
feature_columns = [tf.feature_column.
    numeric_column("x", shape=[1])]
```

TensorFlow for Machine Learning

- The following lines of code show the process of inputting data from a numpy array and model creation.

```
x_train = np.array([1., 2., 3., 4.])
y_train = np.array([0., -1., -2., -3.])
x_eval = np.array([2., 5., 8., 1.])
y_eval = np.array([-1.01, -4.1, -7, 0.])
input_fn = tf.estimator.inputs.numpy_input_fn(
{"x": x_train}, y_train, batch_size=4,
    num_epochs=None, shuffle=True)
train_input_fn = tf.estimator.inputs.
    numpy_input_fn(
{"x": x_train}, y_train, batch_size=4,
    num_epochs=1000, shuffle=False)
eval_input_fn = tf.estimator.inputs.
    numpy_input_fn(
{"x": x_eval}, y_eval, batch_size=4,
```

TensorFlow for Machine Learning

- The next set of lines show training of model and evaluation on test data.

```
# train
estimator.train(input_fn=input_fn, steps=1000)
# Here we evaluate how well our model did.
train_metrics = estimator.evaluate(input_fn=
    train_input_fn)
eval_metrics = estimator.evaluate(input_fn=
    eval_input_fn)
print("train metrics: %r"% train_metrics)
print("eval metrics: %r"% eval_metrics)
```

Deep Learning using TF

We will apply deep learning using TF on the handwritten digit identification dataset - MNIST

```
from tensorflow.examples.tutorials.mnist
    import input_data
mnist = input_data.read_data_sets('MNIST_data'
    , one_hot=True)
```

- We import the dataset and perform one-hot-encoding
- We will launch an interactive session using TF:

```
import tensorflow as tf
sess = tf.InteractiveSession()
```

Deep Learning using TF

- Let's create placeholders for images and labels

```
x = tf.placeholder(tf.float32, shape=[None
    , 784])
y_ = tf.placeholder(tf.float32, shape=[
    None, 10])
```

Note that the dimensions of the image are $28 \times 28 = 784$, that's why x has been defined to have size 784.
Similarly, there are 10 classes, that's why the size of y is 10.

- Let's define weight and bias terms, and also initialize the session.

```
W = tf.Variable(tf.zeros([784,10]))
b = tf.Variable(tf.zeros([10]))
sess.run(tf.global_variables_initializer()
```

Deep Learning using TF

- Initialize the variables and implement the model

```
sess.run(tf.global_variables_initializer()
    )
y = tf.matmul(x,W) + b
```

- Our loss function will the cross entropy:

```
cross_entropy = tf.reduce_mean(
tf.nn.softmax_cross_entropy_with_logits(
    labels=y_, logits=y))
```

- Define training step and run model:

```
train_step = tf.train.
    GradientDescentOptimizer(0.5).minimize(
    cross_entropy)
for _ in range(1000):
        batch = mnist.train.next_batch
            (100)
        train_step.run(feed_dict={x: batch
```

Model Evaluation

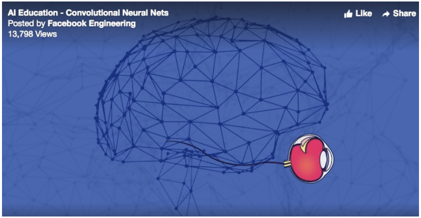
- Let's see how our model did:

```
correct_prediction = tf.equal(tf.argmax(y
    ,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(
    correct_prediction, tf.float32))
print(accuracy.eval(feed_dict={x: mnist.
    test.images, y_: mnist.test.labels}))
```

- Accuracy is around 92%.
  Is it good enough?
- We can do better if we use a Convolution Network - specialized neural net for images and datasets involving multiple representations.

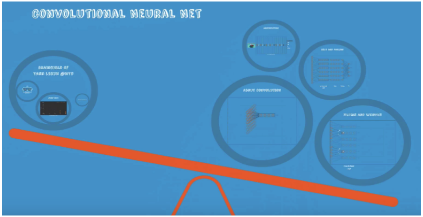
Convolution Neural Network

- Facebook video on CNN



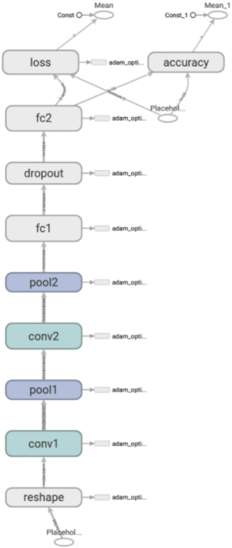
AI Education - Convolutional Neural Nets
Posted by Facebook Engineering
13,798 Views

- Another video:



Convolution Neural Network

- Code for CNN can be viewed here.
- See how much increased accuracy you can get with a CNN
- CNN are shaping the future of learning and can give great results, especially with images.


Mean
Const
Mean_1
Const_1
loss
adam_opti...
accuracy
Placehol...
fc2
adam_opti...
dropout
adam_opti...
fc1
adam_opti...
pool2
adam_opti...
conv2
adam_opti...
pool1
adam_opti...
conv1
adam_opti...
reshape
adam_opti...
Placehol...