



Machine Learning Frameworks and DL

[https://github.com/thenileriver/DH Workshop](https://github.com/thenileriver/DH_Workshop)



Types of Modeling

Supervised learning: training a model using labeled data

- A model trying to classify spam vs. non-spam emails could be trained on labeled emails

Unsupervised learning: training a model using unlabeled data

- A model trying to classify different vegetables could be trained on unlabeled images;

Semi-supervised learning: training a model using both labeled and unlabeled data

Reinforcement learning: utilizes rewards and punishments; actions by the model that work towards the ultimate goal are rewarded

- A model learning to play chess, winning moves are rewarded, losing moves are punished

Deep learning: learning based on artificial neural networks (ANNs); a neural network is considered “deep” when it has multiple layers




Scikit-Learn

- Supervised and unsupervised learning
- Built on NumPy, SciPy, and matplotlib
- Easy to use and offers a high degree of flexibility
- Uses various algorithms and modules to assist mainly with classification, regression, clustering, dimensionality reduction, model selection, and preprocessing
- Integrates well with other ML tools and libraries

Cons of Sklearn

- Lack of deep neural networks
- Lower capabilities of categorical variable transformation
- No support for GPU computing
- Runs slow on large datasets



Scikit-Learn Learning Algorithms

Supervised

- Linear regression
- Logistic regression
- Decision trees
- Random forests
- Support vector machines (SVM)
- Naive Bayes classifiers
- k-Nearest Neighbors (knn)
- Neural networks
- Gradient boosting machines (GBM)
- AdaBoost
- Bagging
- Gradient Descent
- Ridge and Lasso Regression
- Elastic net

Unsupervised

- Clustering
- Dimensionality reduction
- Anomaly detection
- Density estimation

Basic functions example

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
classifier_kNN = KNeighborsClassifier(n_neighbors = 3)
```

Imports kNN classifier algorithm

Initializes kNN object with n_neighbors=3 to consider three nearest neighbors when making predictions

```
classifier_kNN.fit(X_traindata, Y_traindata)
Y_prediction = classifier_kNN.predict(X_testdata)
```

.fit trains the classifier with training data

.predict is applied to test data and obtains predicted labels

```
print("Accuracy of the model is :", metrics.accuracy_score(Y_testdata,
Y_prediction))
```

accuracy_score compares predicted labels with actual labels

```
#We provide sample data for the prediction of KNN
Dummy_data = [[5, 5, 3,2], [4, 3, 6,3]]
pred_value = classifier_kNN.predict(Dummy_data)
prediction_species =[my_iris.target_names[q] for q in pred_value]

print("Our model prediction is :", prediction_species)
```

Sample data for prediction

Makes predictions on Dummy_data with the trained kNN classifier

target_names maps predicted values to species names

Prints predictions

If data is not manually split, use train_test_split() to split data

```
from sklearn.model_selection import train_test_split
X_traindata, X_testdata, Y_traindata, Y_testdata = train_test_split(

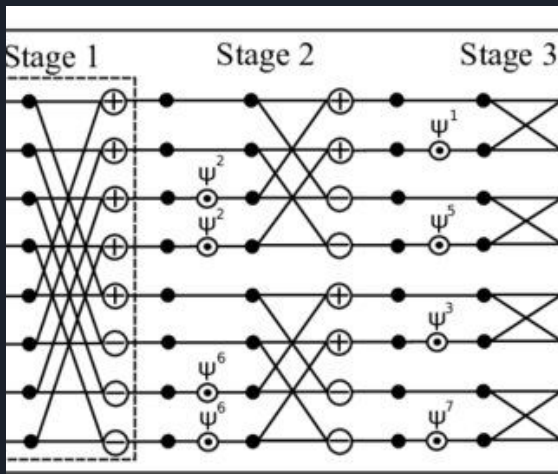
    X, Y, test_size = 0.3, random_state = 1)
```

Splits data into train datasets (used to fit the model) and test datasets (evaluates model's performance)

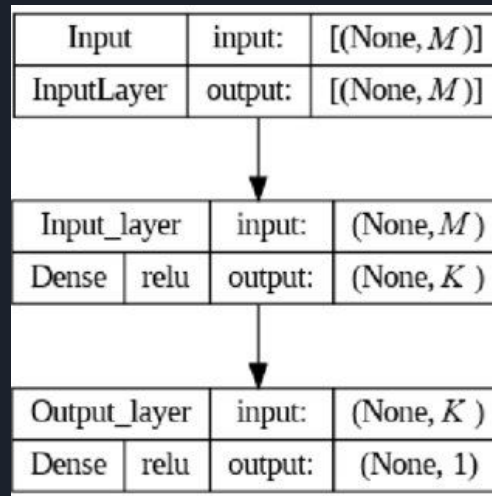
Splits 70% for training and 30% for test data

Keras: Neural Networks and Deep Learning

- High-level, deep learning API built on TensorFlow (Python), as well as other deep learning libraries
 - Tensorflow functionality is based on computational graphs



- Keras allows for the creation of neural layers
- Neural layers can be paired to make custom models
- For example, a sample deep learning algorithm model:



A Sample Neural Network with Keras

```
model.add(Dense(5, kernel_initializer = 'uniform', activation = 'relu', input_dim = 5))
model.add(Dense(5, kernel_initializer = 'uniform', activation = 'relu'))
model.add(Dense(1, kernel_initializer = 'uniform', activation = 'sigmoid'))
```

- This is a Sequential model made in Keras made to predict whether a person would survive the Titanic
 - It has 3 layers:
 - The input layer has 5 dimensions, aka 5 inputs
 - The middle layer does all the processing to determine a result
 - And the final layer returns a result as an integer from (0,1)

What the model looks like:

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 5)	30
dense_1 (Dense)	(None, 5)	30
dense_2 (Dense)	(None, 1)	6
Total params: 66		
Trainable params: 66		
Non-trainable params: 0		



Keras's Uses

Pros:

- Makes deep learning more approachable and easier to use
- Performs well on large-scale data and deep learning problems

Cons:

- Tends to be slow at executing and training
- Takes a lot of computing and processing resources
- Not as efficient for smaller-scale data/non-deep learning problems



Application to Neural Net Architecture

Working with fossil images?

- Try using a CNN like U-Net or ResNet

Working with time forecasting (climate, extinction, etc.)

- RNNs are great for time series based data

Creating data for voided species

- Use GANs to create data on species within the same family to then predict off of

Predict the noise a dinosaur makes

- Use an autoencoder to predict the noise a dino makes by cross referencing the noise their evolved species makes



Additional Resources

Pytorch (Patrick Loeber's Playlist):

<https://www.youtube.com/watch?v=EMXfZB8FVUA&list=PLqnsIRFeH2UrcDBWF5mfPGpqQDSta6VK4>

An Amazing Repo: <https://github.com/ChristosChristofidis/awesome-deep-learning>

Paleobiology Database: <https://paleobiodb.org/#/>