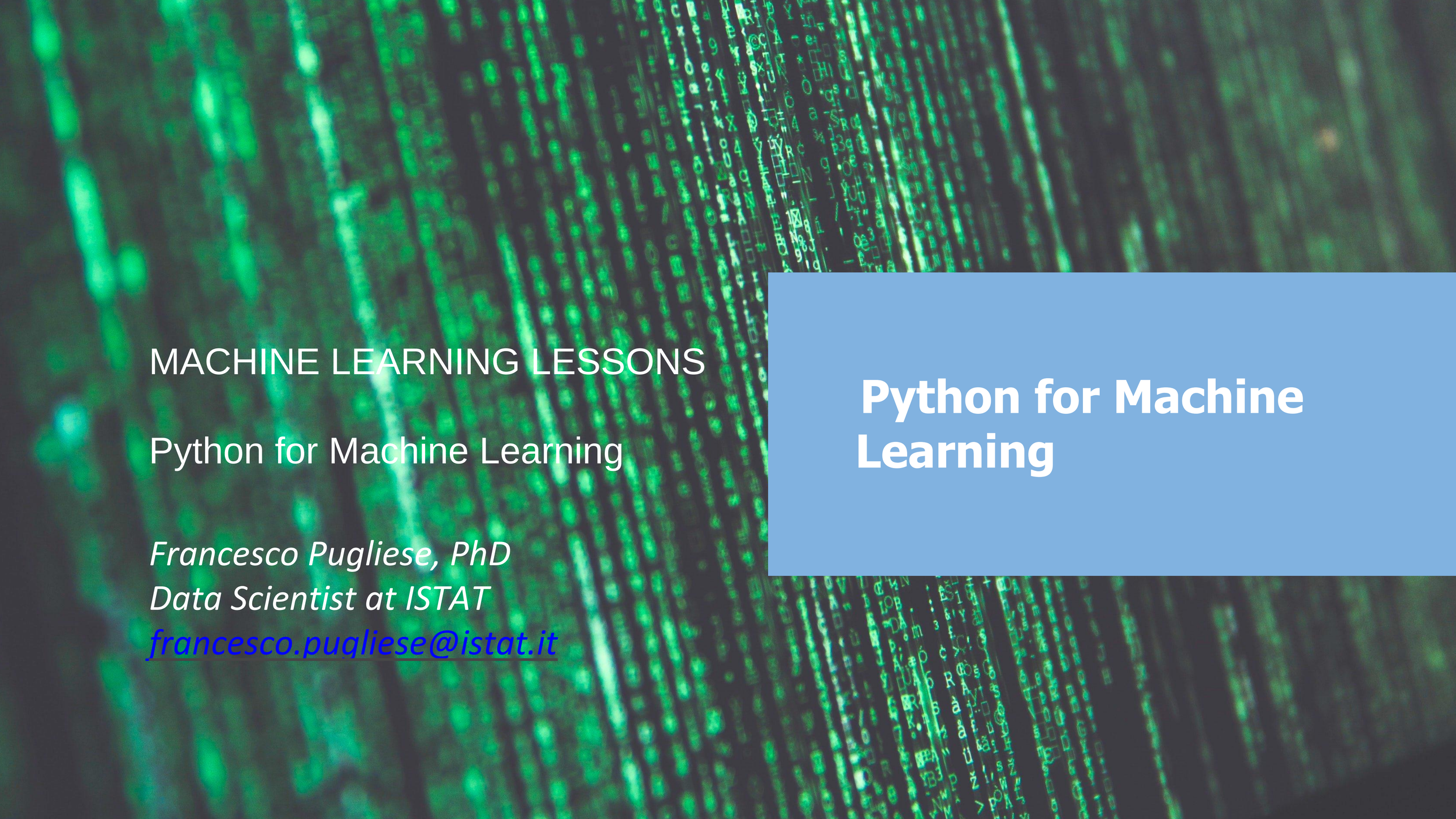


MACHINE LEARNING LESSONS

Python for Machine Learning

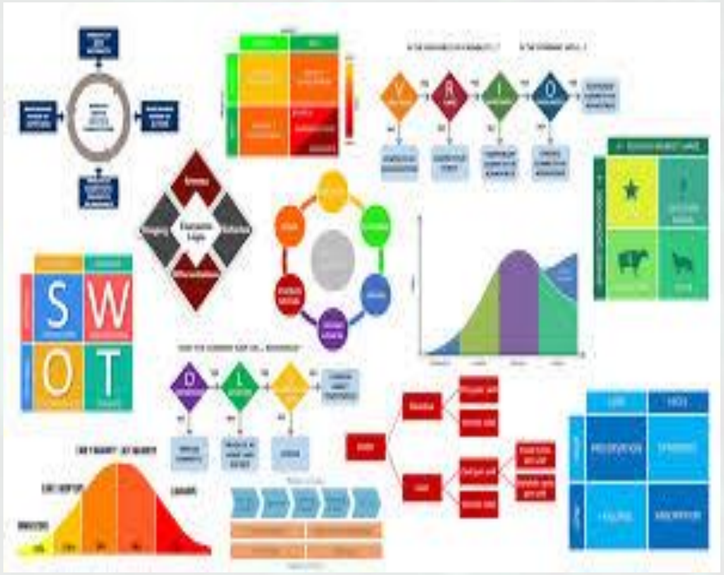
Francesco Pugliese, PhD
Data Scientist at ISTAT
francesco.pugliese@istat.it

Python for Machine Learning

Why starting with Python?

- **Python** is a perfect choice for beginner to make your focus on in order to jump into the field of machine learning and data science. It is a minimalistic and intuitive language with a full-featured library line (also called frameworks) which significantly reduces the time required to get your first results.

Frameworks for Deep Learning

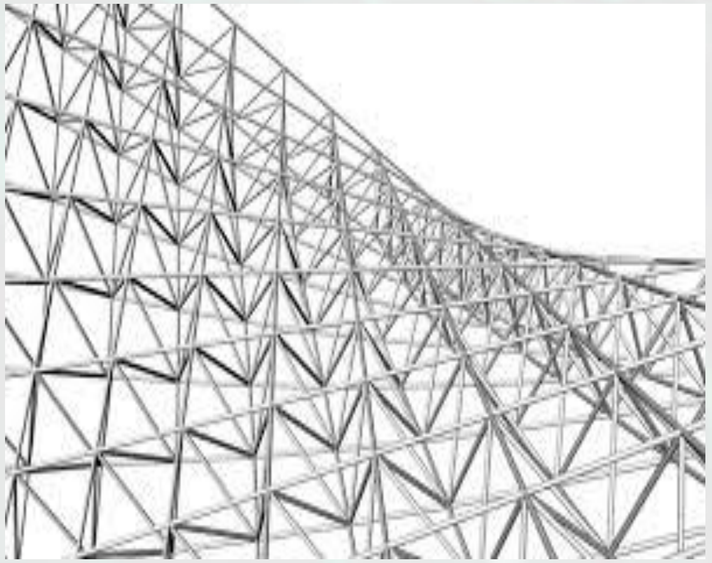


Keras is an higher-level interface for Theano (which works as backend). Keras displays a more intuitive set of abstractions that make it easy to configure neural networks regardless of the backend scientific computing library.

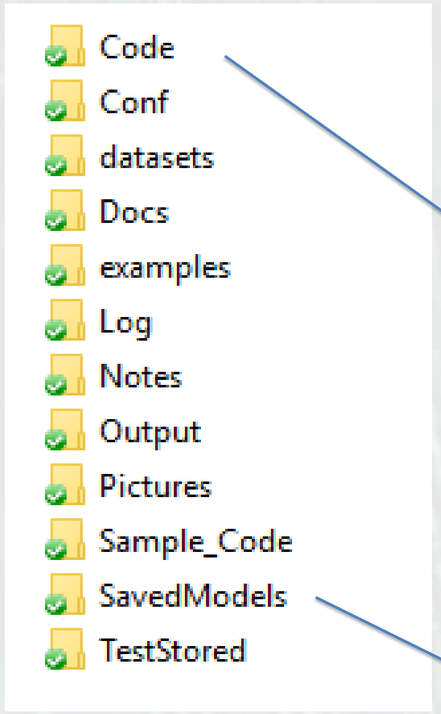
TensorFlow is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library, and also used for machine learning applications such as neural networks. It is used for both research \and production at Google.

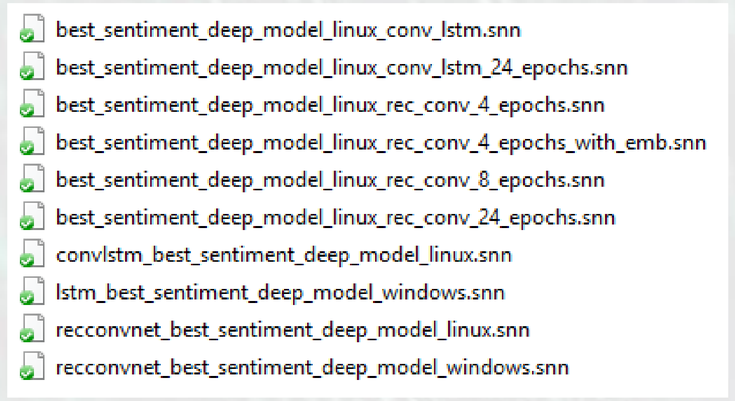
PyTorch is an open-source machine learning library for Python, derived from Torch, used for applications such as natural language processing. It is primarily developed by Facebook’s artificial-intelligence research group, and Uber’s "Pyro" software for probabilistic programming is built on it.

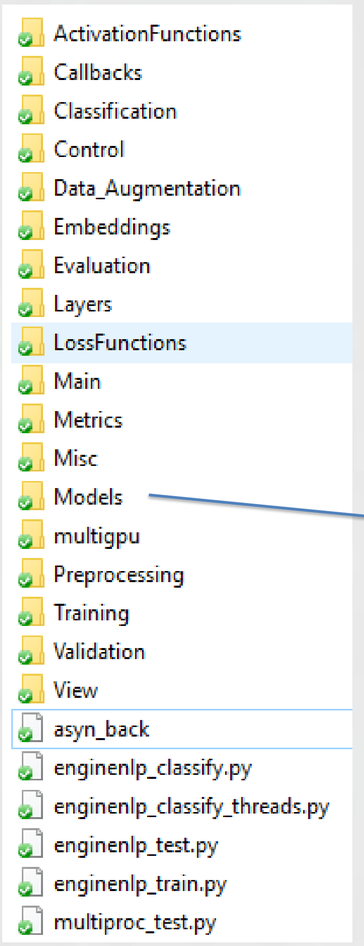
Anatomy of an advanced Python Machine Learning Application

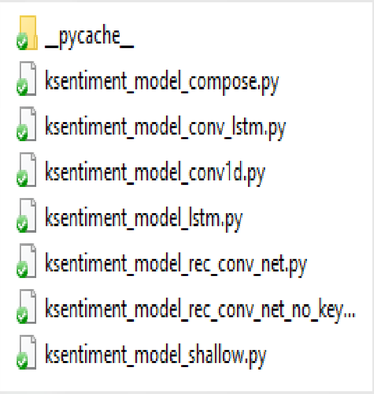


Python for Machine Learning
Francesco Pugliese









Keras: The Python Deep Learning Library

- **Keras** is an higher-level API to build neural networks Quickly. It is written in Python and works in Python.
- **Keras** works on top of **Tensorflow, CNTK** or **Theano** accordingly. It was developed keeping in mind prototyping and fast experimentation.
- **Keras** is able to go from the idea to the result with the least possible delay, which is the key to doing both good research and value business.
- We can adopt **Keras** in the cases we need:

  1) Enabling fast prototyping by means of friendliness, modularity and extensibility
  2) Supporting both Cconvolutional Networks and Recurrent Networks, as well as combinations of the two.
  3) Running seamlessly on **CPU** and **GPU**

Keras: The Python Deep Learning Library

READ THE DOCUMENTATION AT keras.io

Keras is compatible with Python 2.7 / 3.6.

- The core data structure of **Keras** is a model, namely a way to manage layers easily. The simplest type of model can be implemented by the class ***Sequential***, which enables to build a model as a linear stack of layers.

- For more complex topologies of neural networks, implementing parallel layers for instance, we need the ***Keras Functional API***, enabling to build arbitrary graphs of layers

- Example of implementation of **Sequential Model:**

```
from keras.models
model = Sequential()
```

Getting started: 30 seconds to Keras Library

- Stacking layers is easy by the method .add of the class Sequential.

```
from keras.layers import Dense
model = Sequential()
model.add(Dense(units = 64, activation = ‘relu’, input_dim = 100))
model.add(Dense(units = 10, activation = ‘softmax’)
```

- After the model definition, it is necessary to configure the the learning process with the method **.compile**

```
model.compile(loss=‘categorical_crossentropy’,
              optimizer=‘sgd’, metrics = [‘accuracy’])
```

Getting started: 30 seconds to Keras Library

- After compiling the neural model it is possible to iterate on training data in batches by using **.fit** method.

```
model.fit(x_train, y_train, epochs = 5, batch_size = 32)
```

- Alternatively, we can train the model on each batch manually, by means of **.train_on_batch**.

```
model.train_on_batch(x_batch, y_batch)
```

- Then we need to evaluate our model on a test set for example. model on each batch manually

```
model.train_on_batch(x_test, y_tes, batch_size = 128)
```


- **NOTE:** in case of evaluation or prediction the batch size can be any size, and it does not the accuracy of the model such as in the training stage. In test phase batch_size depends exclusively on the size of GPU. Batch_size hyperparameter so can scale up the test stage according to the computational capabilities.

Getting started: 30 seconds to Keras Library

- After compiling the neural model it is possible to iterate on training data in batches by using **.fit** method.

```
model.fit(x_train, y_train, epochs = 5, batch_size = 32)
```

- Alternatively, we can train the model on each batch manually, by means of **.train_on_batch**.

```
model.train_on_batch(x_batch, y_batch)
```

- Then we need to evaluate our model performance on a test set for example.

```
loss_and_metrics = model.evaluate(x_test, y_test, batch_size = 128)
```

- **NOTE:** in case of evaluation or prediction the batch size can be any size, and it does not the accuracy of the model such as in the training stage. In test phase batch_size depends exclusively on the size of GPU. Batch_size hyperparameter so can scale up the test stage according to the computational capabilities.

Getting started: 30 seconds to Keras Library

- At this point, we can adopt the trained model in order to predict categorical variables (classification) or continuous variables (regression). For this purpose, we need the ***.predict*** method of the class **Sequential**.

```
classes = model.predict(x_test, batch_size = 128)
```

- This way, it is simple and fast to build a neural chatbot or a question-answer system, an image classification model, a text classifier or any other model.

- The idea behind keras is that Deep Learning implementations are not painful anymore.

Keras Sequential Model

The `Sequential` model is a linear stack of layers.

You can create a `Sequential` model by passing a list of layer instances to the constructor:

```
from keras.models import Sequential
from keras.layers import Dense, Activation

model = Sequential([
    Dense(32, input_shape=(784,)),
    Activation('relu'),
    Dense(10),
    Activation('softmax'),
])
```

You can also simply add layers via the `.add()` method:

```
model = Sequential()
model.add(Dense(32, input_dim=784))
model.add(Activation('relu'))
```

Keras Sequential Model

Python for Machine Learning

Francesco Pugliese

Compilation

Before training a model, you need to configure the learning process, which is done via the `compile` method. It receives three arguments:

- An optimizer. This could be the string identifier of an existing optimizer (such as `rmsprop` or `adagrad`), or an instance of the `Optimizer` class. See: optimizers.
- A loss function. This is the objective that the model will try to minimize. It can be the string identifier of an existing loss function (such as `categorical_crossentropy` or `mse`), or it can be an objective function. See: losses.
- A list of metrics. For any classification problem you will want to set this to `metrics=['accuracy']`. A metric could be the string identifier of an existing metric or a custom metric function.

```
# For a multi-class classification problem
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# For a binary classification problem
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# For a mean squared error regression problem
model.compile(optimizer='rmsprop',
              loss='mse')

# For custom metrics
import keras.backend as K

def mean_pred(y_true, y_pred):
    return K.mean(y_pred)

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy', mean_pred])
```

Keras Sequential Model

Python for Machine Learning

Francesco Pugliese

Training

Keras models are trained on Numpy arrays of input data and labels. For training a model, you will typically use the `fit` function. Read its documentation here.

```
# For a single-input model with 2 classes (binary classification):

model = Sequential()
model.add(Dense(32, activation='relu', input_dim=100))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Generate dummy data
import numpy as np
data = np.random.random((1000, 100))
labels = np.random.randint(2, size=(1000, 1))

# Train the model, iterating on the data in batches of 32 samples
model.fit(data, labels, epochs=10, batch_size=32)
```

```
# For a single-input model with 10 classes (categorical classification):

model = Sequential()
model.add(Dense(32, activation='relu', input_dim=100))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Generate dummy data
import numpy as np
data = np.random.random((1000, 100))
labels = np.random.randint(10, size=(1000, 1))

# Convert labels to categorical one-hot encoding
one_hot_labels = keras.utils.to_categorical(labels, num_classes=10)

# Train the model, iterating on the data in batches of 32 samples
model.fit(data, one_hot_labels, epochs=10, batch_size=32)
```

Keras Sequential Model

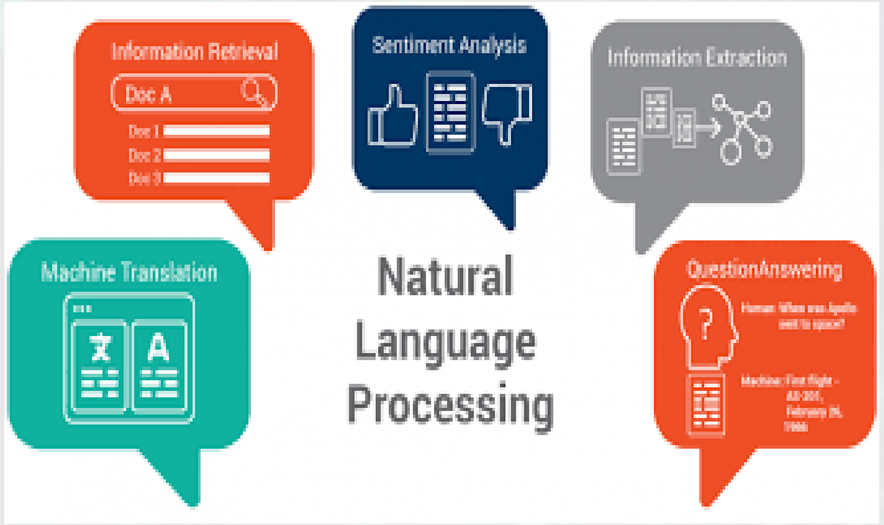
Python for Machine Learning

Francesco Pugliese

Multilayer Perceptron (MLP) for multi-class softmax classification:

```
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from keras.optimizers import SGD

# Generate dummy data
import numpy as np
x_train = np.random.random((1000, 20))
y_train = keras.utils.to_categorical(np.random.randint(10, size=(1000, 1)), num_classes=10)
x_test = np.random.random((100, 20))
y_test = keras.utils.to_categorical(np.random.randint(10, size=(100, 1)), num_classes=10)

model = Sequential()
# Dense(64) is a fully-connected layer with 64 hidden units.
# in the first layer, you must specify the expected input data shape:
# here, 20-dimensional vectors.
model.add(Dense(64, activation='relu', input_dim=20))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy',
              optimizer=sgd,
              metrics=['accuracy'])

model.fit(x_train, y_train,
          epochs=20,
          batch_size=128)
score = model.evaluate(x_test, y_test, batch_size=128)
```

Dataset Loading




```
texts = []  # list of text samples
labels_index = {}  # dictionary mapping label name to numeric id
labels = []  # list of label ids
for name in sorted(os.listdir(TEXT_DATA_DIR)):
    path = os.path.join(TEXT_DATA_DIR, name)
    if os.path.isdir(path):
        label_id = len(labels_index)
        labels_index[name] = label_id
        for fname in sorted(os.listdir(path)):
            if fname.isdigit():
                fpath = os.path.join(path, fname)
                if sys.version_info < (3,):
                    f = open(fpath)
                else:
                    f = open(fpath, encoding='latin-1')
                t = f.read()
                i = t.find('\n\n')  # skip header
                if 0 < i:
                    t = t[i:]
                texts.append(t)
                f.close()
                labels.append(label_id)

print('Found %s texts.' % len(texts))
```

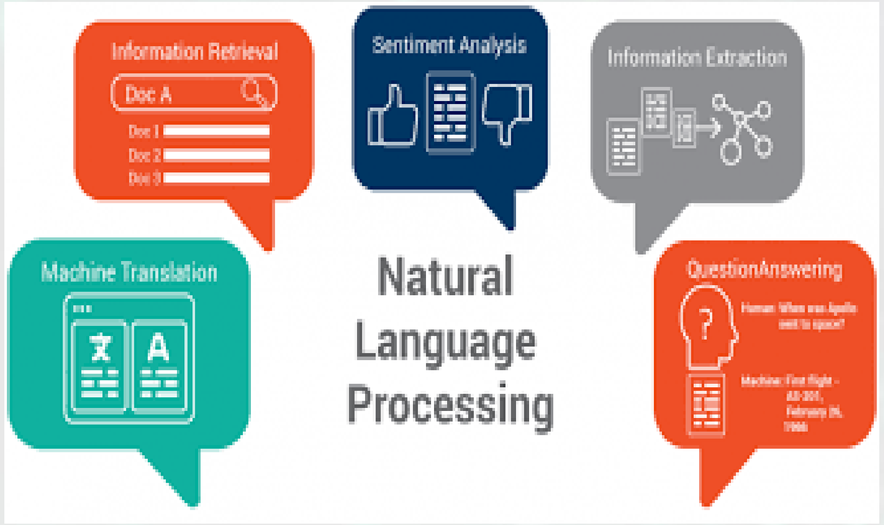
ARCHIVED: What is the Latin-1 (ISO-8859-1) character set?

This content has been archived, and is no longer maintained by Indiana University. Resources linked from this page may no longer be available or reliable.

Latin-1, also called ISO-8859-1, is an 8-bit character set endorsed by the International Organization for Standardization (ISO) and represents the alphabets of Western European languages. As its name implies, it is a subset of ISO-8859, which includes several other related sets for writing systems like Cyrillic, Hebrew, and Arabic. It is used by most Unix systems as well as Windows. DOS and Mac OS, however, use their own sets.

Building Texts List and Labels List

Dataset Loading




```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

tokenizer = Tokenizer(nb_words=MAX_NB_WORDS)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
```

Stop-words Cleaning and tokenization

```
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
```

Building of the Word Index

```
data = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH)
```

Sequence 1D 0-Padding

```
labels = to_categorical(np.asarray(labels))
print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', labels.shape)

# split the data into a training set and a validation set
indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]
nb_validation_samples = int(VALIDATION_SPLIT * data.shape[0])

x_train = data[:-nb_validation_samples]
y_train = labels[:-nb_validation_samples]
x_val = data[-nb_validation_samples:]
y_val = labels[-nb_validation_samples:]
```

To categorical vari ables conversion

Shuffle of Data an d Labels

Data and Labels S plitting

Natural Language Toolkit

Natural Language Toolkit

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.

Thanks to a hands-on guide introducing programming fundamentals alongside topics in computational linguistics, plus comprehensive API documentation, NLTK is suitable for linguists, engineers, students, educators, researchers, and industry users alike. NLTK is available for Windows, Mac OS X, and Linux. Best of all, NLTK is a free, open source, community-driven project.

NLTK has been called "a wonderful tool for teaching, and working in, computational linguistics using Python," and "an amazing library to play with natural language."

Natural Language Processing with Python provides a practical introduction to programming for language processing. Written by the creators of NLTK, it guides the reader through the fundamentals of writing Python programs, working with corpora, categorizing text, analyzing linguistic structure, and more. The online version of the book has been been updated for Python 3 and NLTK 3. (The original Python 2 version is still available at http://nltk.org/book_1ed.)

Natural Language Toolkit

Some simple things you can do with NLTK

Tokenize and tag some text:

```
>>> import nltk
>>> sentence = """At eight o'clock on Thursday morning
... Arthur didn't feel very good."""
>>> tokens = nltk.word_tokenize(sentence)
>>> tokens
['At', 'eight', "o'clock", 'on', 'Thursday', 'morning',
'Arthur', 'did', "n't", 'feel', 'very', 'good', '.']
>>> tagged = nltk.pos_tag(tokens)
>>> tagged[0:6]
[('At', 'IN'), ('eight', 'CD'), ("o'clock", 'JJ'), ('on', 'IN'),
('Thursday', 'NNP'), ('morning', 'NN')]
```

Practical Machine Learning Case Study

- The goal of this Case Study is to predict how much a house will sell for. This is for example similar to many problems Amazon has in online advertising and merchandising, where they try to predict what value a customer has for an item.

- In order to reach this objective we have to produce a Machine Learning model and an evaluation metric.

- You can find and download the data at:
  http://bit.ly/1Fq0svx
- A description of the columns in the data is available at:
  http://bit.ly/1hh97JI

The file to download is named ***"HM Land Registry Price Paid Data"***

Practical Machine Learning Case Study

Explanations of column headers in the PPD

The data is published in columns in the order set out in the table, we do not supply column headers in the files.

Data item	Explanation (where appropriate)
Transaction unique identifier	A reference number which is generated automatically recording each published sale. The number is unique and will change each time a sale is recorded.
Price	Sale price stated on the transfer deed.
Date of Transfer	Date when the sale was completed, as stated on the transfer deed.
Postcode	This is the postcode used at the time of the original transaction. Note that postcodes can be reallocated and these changes are not reflected in the Price Paid Dataset.
Property Type	D = Detached, S = Semi-Detached, T = Terraced, F = Flats/Maisonettes, O = Other Note that: - we only record the above categories to describe property type, we do not separately identify bungalows. - end-of-terrace properties are included in the Terraced category above. - 'Other' is only valid where the transaction relates to a property type that is not covered by existing values.
Old/New	Indicates the age of the property and applies to all price paid transactions, residential and non-residential. Y = a newly built property, N = an established residential building

Practical Machine Learning Case Study

Duration	Relates to the tenure: F = Freehold, L= Leasehold etc. Note that HM Land Registry does not record leases of 7 years or less in the Price Paid Dataset.
PAON	Primary Addressable Object Name. Typically the house number or name.
SAON	Secondary Addressable Object Name. Where a property has been divided into separate units (for example, flats), the PAON (above) will identify the building and a SAON will be specified that identifies the separate unit/flat.
Street	
Locality	
Town/City	
District	
County	

Practical Machine Learning Case Study

Columns of interest for our task are:

- **Column 1** contains a unique ID for the purchase and can be ignored.
- **Column 3** contains the date of the purchase
- **Column 5** contains the property type
- **Column 7** contains the lease duration
- **Column 12** contains the city or the town where the property was located. You can judge a property to London if the field contains the word “London”.

MACHINE LEARNING LESSONS

Python for Machine Learning

Francesco Pugliese, PhD
Data Scientist at ISTAT
francesco.pugliese@istat.it

Thank You