The ML LabBook

Principal Contributors:

Rajat Agarwal
Sanket Gupte

Principal Editors:

Ashwin Srinivasan
Tirtharaj Dash

Department of Computer Science & Information Systems
BITS Pilani Goa Campus
Goa, India 403726

Contents

1. Tools

1.1. Programs

1.1.1. Python

Python[1] is a high level general purpose programming language, with an easy to learn syntax. It is widely used for numerous tasks ranging from Web Development to Big Data Analysis. Python is an interpreted language, making it useful for rapid prototyping and testing ideas. It also has an extensive standard library of modules to provide functionality for more specialized tasks such as socket programming or XML parsing. A rich ecosystem of third party modules and libraries for almost all other tasks makes Python a language of choice for many developers.

Although Python's advantages like dynamic interpretation and automatic type inference make it slow during execution as compared to languages like C, it is still widely used for computationally intensive tasks such as Machine Learning (ML) because the performance critical code can be written in fast languages like C and Fortran, thus giving programmers the ability to leverage Python's flexibility without sacrificing performance.

The Python Interpreter

If you are using Unix/Linux based machine, you can invoke the Python Interpreter from the command line by typing the command "python". If you are using Windows machine, then you could use a GUI based application called IDLE (https://docs.python.org/3.6/library/idle.html) This will start a session where you can type python commands to execute them one by one. You will also see information such as the version number and compiler version. It is recommended to use v3.X for current projects. Note - many Linux systems have both versions 2.X and 3.X installed. You can use the command "python3" to launch the 3.X version

Running Python programs

In addition to being run in the interpreter, you can also write code and save it as a .py file. This is the preferred way to run code that is more than a few lines long, or can be executed repeatedly. Use the Python shell only to quickly test code, for example to check if a package is installed by trying to import it, or to test an expression on a few examples. For all other cases, you can use files with the .py extension and execute them by typing "python <filename>.py". As with other programming languages, you can also provide command line arguments to your code.

Installing third party Libraries

The vast majority of third party libraries have already been published on the Python Package Index (PyPI)[2]. You can search this repository for packages that you need. Many of these will provide you with the source code that you can download and install by navigating to the folder and running the setup file with "python setup.py install". You should note that you may require administrative privileges to install the downloaded files. However, a better way of installing files is by using the 'pip installer'. The pip

[1] https://www.python.org/
[2] https://pypi.python.org/pypi

installer makes it easy to download and install modules and their dependencies with a single command. For example, installing TensorFlow is as easy as typing “pip install tensorflow”.

Exercise 1.1.1.a. Go through the official Python documentation at https://docs.python.org/3/ and complete the tutorial and read the library references, especially the sections pertaining to the concepts below.

Important Python Concepts

If you’re using Python for ML, there are a few core concepts that you need to be aware of, which will help you write faster, better, and more concise codes.

A. Familiarity with Python Data Structures and their runtime complexities

Python has a number of built in data containers such as the List, Set, Dictionary, and Tuple. In addition to these, there are a number of high performance containers in the standard library such as deques, heaps etc. Understanding the complexity of insertions, deletions and searches of these containers will help you choose the right one for the task at hand.

B. Object Oriented Programming with Python

Python supports Object Oriented Programming (OOP) paradigms out of the box. Although you may not write your own classes and interfaces very often (unlike in Java), most of the code in external libraries is written in this fashion to support modularity and code reuse. Familiarity with such principles will help you use such code more efficiently.

C. String manipulation and File I/O

A major part of the ML workflow involves data processing. Python makes this easy with its powerful String manipulation functions, the file reading functions, and the iterators. Knowing how to use regular expressions will help you parse and filter string data. Here is an example of reading a CSV file and saving it as a 2D array (list of lists):

```
data = [line.strip().split(‘,’) for line in open(‘my_file.csv’)]
```

D. Functional Programming paradigms such as Map, Reduce, and Filter

Map, Reduce, and Filter are powerful functions when used in conjunction with lambdas.
Squaring all elements of a list:

```
squared_data = map(lambda x: x*x, data)
```

Removing Nones from a list:

```
filtered_data = filter(lambda x: x is not None, data)
```

Product of all elements of a list:

```
prod = reduce(lambda x,y: x*y, data)
```

E. Parallel Programming

Python provides support for parallel execution of code on multiple cores through the multi-processing module. Although limited, this support is good for accelerating Map tasks for a large input, e.g. doing data processing on thousands of files, training dozens of ML models, and suchlike.

1.1.2. Numpy

NumPy[3] is a popular Numerical Python data processing library. It is primarily written in C and Fortran to provide very fast and efficient data manipulation routines. It has the ability to scale certain functions across multiples cores. [a1]The python wrapper allows programmers to leverage this speed without sacrificing flexibility. NumPy is one of the foundation packages for most ML, DL (Deep Learning) and scientific computing libraries. Due to this reason, it serves as one of the most crucial prerequisite for getting started with ML.

NumPy's core object is the N-Dimensional Array (also called NDarray/ndarrays), which is homogeneous in nature (single datatype). The number of axes in the array is called the 'rank' of the array, and it is indexed via non-negative integers. A vector would have rank 1 and a matrix has rank 2. These arrays support operations with binary operators, slicing, searching and sorting, reshaping, copying, and some linear algebra routines.

Basics

Array Attributes: ndim–rank of the array, shape–tuple of dimensions, dtype–Data Type of the object

```
Creating an Array :          a = np.array([1,2,3,4,5])
Specifying datatype :        a = np.array([1,2,3,4,5], dtype = np.float32)
Creating a matrix of zeros:  a = np.zeros((rows, cols))
Specifying a range:          a = np.arange(0,1,0.1)

Creating a sine wave:        x = np.linspace(0, 2*np.pi, 100)      #100 points between 0-2pi
                             sin_x = np.sin(x)
```

In the above examples, 'np' is a short name for numpy. To use any library function, you must import the header library (similar to header files in C) to use it's routines. In this case, you should write:

```
import numpy as np
```

You can also import a set of defined routines instead of the full library asfrom math import sqrt, sin

Operators

```
Arithmetic operators         z = x + y      # x and y are ndarrays
                             z = x * y      # y can also be a scalar
                             z = x / y      # All operations are done element wise

Comparison operators         z = x > y      # returns a Boolean array of element wise results
                             z = x > 5      # comparing x with 5 element wise

Logical and Bitwise operators  z = np.logical_and(x, y)
```

[3] http://www.numpy.org/

```
z = np.bitwise_and(x, y)
z = x & y          # equivalent to bitwise_and
z = x | y          # bitwise_or. Don't confuse it with logical_or
```

Unary operators — These operators take and axis optionally along which to perform the op

```
mean  = np.mean(A)
mean = A.mean()          # equivalent to the previous one
total = a.sum(axis = 0)  # calculates sum of each column
std_dev = a.std(axis = 1) # calculates standard deviation for each row
```

Indexing and Slicing

NumPy arrays can be indexed and sliced just like lists. However, multi-dimensional arrays can be indexed and sliced on each given axis, using comma(s) to separate the slicing range for each axis.

```
x = np.arange(0,5,1)     # [0 1 2 3 4]
x[2:4]                   # [2 3]       slice a range
x[2:4] = 5               # [0 1 5 5 4]     set a range
x[:-2]                   # [0 1 2]     slice up to last 2 elements

x = np.zeros((3,3))      # [[0 0 0], [0 0 0], [0 0 0]]      3x3 zero array
x[0] = 1                 # [[1 1 1], [0 0 0], [0 0 0]]
x[0]                     # [1 1 1]
x[1,2] = 3               # [[1 1 1], [0 0 3], [0 0 0]]      row 1, column 2 is set to 3
x[:,2]                   # [1 3 0]                          last column
x[0:2, 1:3]              # [[1 1], [0 3]]
x[x>0]                   # [1 1 1 3]                        fancy indexing with Boolean arrays
```

Reshaping

```
a = np.zeros(24)# shape is (24,)
a = a.reshape(1, 24)     # a now has 1 row, 24 columns
a = a.reshape(24,1)      # a now has 24 rows, 1 column
a = a.reshape(2,3,4)     # a now has 2 3x4 matrices
a = a.flatten()          # shape is (24,)
a = a.reshape(-1,2,3)    # missing dimension (-1) is deduced automatically to give (4,2,3)
```

Note: Data is usually stored in memory in contiguous locations. Reshaping does not change this layout, and merely redefines the boundaries for each axis. If you want to change the axes, use transpose.

Linear Algebra Operations

```
a = a.transpose(1,2,0)          # changes an array of (channels, rows, cols) to (rows, cols, channels)
a = a.T                         # transpose shortcut for 2D arrays. (rows, cols) becomes (cols,rows)
a = np.linalg.inv(a)            # finds inverse of a square matrix
x = a.dot(b)      # computes dot product of a and b. Equivalent to matrix multiplication for 2D arrays.
```

Exercise 1.1.2.a. Use the linear algebra operations to solve for R in the equation, P=QR, where P.shape = (N, 1), Q.shape = (N,M), and R.shape = (M,1). Verify your solution by checking if P is equal to QR. You can generate random arrays of any shape using the np.random.random(<shape_tuple>) function.

1.1.3. Scipy

SciPy[4] is an open-source software for scientific computing and covers the disciplines of mathematics, science and engineering. It is built on top of NumPy and relies heavily on its ndarray data structure for storing information. It provides extensions through routines developed for a range of tasks such as Integration, Optimization, Interpolation, Fourier Transforms, Signal Processing, Linear Algebra, Statistics and basic Image Processing.

Although the general purpose nature of SciPy means that there are some modules which you may not use for everyday ML, the following are essential for performing basic tasks

Optimization - The scipy.optimize module contains support for several types of optimization problems such as unconstrained minimization of multivariate scalar functions, constrained minimization, least-squares minimization and root finding.

Linear Algebra - The scipy.linalg module is similar to the numpy.linalg module but has some more advanced features, such as solving linear equations, solving linear least-squares problems and pseudo inverses, and matrix decomposition methods such as Eigenvalue decomposition which is used in Principal Component Analysis

Spatial - The scipy.spatial module provides algorithms and data structures that are designed to operate on the spatial properties of the data. Convex hull finding algorithms can be used to find the best discrimination threshold by analyzing performance curves of ML algorithms. Spatial data structures such as the KD-Tree are especially useful in ML algorithms that rely on calculating distance to neighboring points such as the k Nearest Neighbors algorithm

Statistics - The scipy.stats module has utilities for generating and analyzing various types of statistical distributions, as well as computing metrics such as kurtosis and conducting skew tests, T-tests, computing *p*-values etc. It also has some utilities for applying transforms on data, such as the Box-Cox transform for data having a long tail distribution (which makes it easier to train regression models)

Exercise 1.1.3.a. Generate data points that obey the equation $y = c1e^{-x} + c2x$ for some value of c1 and c2. Now try to estimate these coefficients by solving using x and y values. Verify your solution by comparing LHS and RHS values once you determine the coefficients.

Exercise 1.1.3.b. Add some Gaussian noise to the y values and again try to estimate the coefficients. How much do these values change? What is the mean squared error of the curve you have fit?

Exercise 1.1.3.c. Generate N points of D dimensions. Write Python code to find the 5 nearest neighbors of each point. Compare the time taken to do this with the scipy.spatial.CKDtree implementation and measure the difference.

[4] https://docs.scipy.org/doc/scipy/reference/

1.1.4. Pandas

Pandas[5] is a data storage and analysis library that primarily provides utilities to deal with structured records, normally stored as CSVs or tables. It provides powerful tools to analyze and describe data, and to transform, augment, and filter it. This makes it a popular tool to handle the majority of data processing operations. Pandas library provides the DataFrame class as its core storage structure. Most operations are vectorized and highly optimized to improve performance.

Import the pandas library first:

```
import numpy as np

import pandas as pd
```

Reading CSVs

```
df = pd.read_csv("file_name.csv")
```

Analyzing DataFrames

```
df.head()          # prints the first few rows of the dataframe
df.info()          # provides a concise summary of the dataframe
df.describe()      # provides descriptive statistics of central tendency, dispersion and shape
```

Views and Slicing

```
df["marks"]                          # gets the data from column "marks"
del df["marks"]                      # deletes the marks column
df["passed"] = df["marks"] > 50      # creates a new column with True / False values is marks > 50
df[df.passed]                        # selecting rows where passed == True
df[df.passed][:10]                   # selecting first 10 rows where passed = True
```

Missing Values

```
pd.isnull(df)              # checks for null or missing values
df.fillna(0)               # fills missing values with 0
df.filllna(df.mean())      # fills missing values with the mean value of each column
df.dropna(axis = 0)        # removes rows with missing values
df.dropna(axis = 0)        # removes columns with missing values
```

Adding new columns

```
df = df.assign(   sin = np.sin(df.angle),
                  cos = np.sin(df.angle),
                  tan = np.sin(df.angle))   # add new columns by applying a function on the angle column
```

Exercise 1.1.4.a. Try different types of dataframe joins[6] with pandas. Compare them with SQL table joins.

[5] http://pandas.pydata.org/
[6] https://pandas.pydata.org/pandas-docs/stable/

1.1.5. Scikit-learn

Scikit-Learn[7] is a Python library for high performance Machine Learning. It leverages NumPy and SciPy, along with additional C modules to provide extremely fast performance, while organizing the code in a highly modular object oriented fashion to facilitate ease of use. Its extensive coverage of the most widely used ML algorithms makes it a popular choice for beginners and professionals alike. It provides the following functionality

Supervised Learning

Classification

Classification problems involve predicting one or more class labels for the input, for example predicting if an email is spam or not. Scikit-Learn has a unified classifier API that has the following interface.

sklearn.module.Classifier is the constructor for the classifier object, where you can pass specific parameters (params).

E.g. *sklearn.ensemble.RandomForestClassifier(n_estimators = 100, max_depth = 6)*

Each classifier has a *fit* method that allows it to be trained on some data, consisting of a training set (N_samples, N_features) and the associated labels (N_samples, 1).

Once *fit* has been called, you can make predictions on unseen data using the *predict* function. This returns a 1D array consisting of a single label for each data point. Using *predict* before *fit* will raise an error. Many classifiers support class probability outputs as well through the *predict_proba* function. It returns a score between 0 and 1 for each possible class, for each data point.

Regression

Regression involves predicting continuous outputs for a given set of inputs, for example predicting the price of a house based on several factors such as its layout, area, locality etc. The interface is similar to that of the classifier API, with a few differences.

fit can be used to build multiple regressors at once by passing in targets of shape (N_samples, N_outputs)

predict outputs an array of continuous values, and hence is an array of floats, rather than an array of integers as with classification. There is no *predict_proba* function.

Unsupervised Learning

Clustering

Clustering of unlabeled data is the task of grouping data points together in such a way that points in a cluster are similar to each other. It is useful for discovering patterns in the distribution of the data, and is hence used during exploratory analysis.

All algorithms are present in the sklearn.cluster module and have the following interface.

[7] http://scikit-learn.org/stable/index.html

sklearn.cluster.ClusteringMethod is the constructor, and some of these methods require you to specify the number of clusters during initialization, while others automatically discover them. *fit* and *predict* functions work the same way as with Classifiers and Regressors, but since this is unsupervised learning, labels are not passed to the *fit* function.

Dimensionality Reduction

Dimensionality Reduction is the process of reducing the number of input variables while attempting to preserve essential information. Algorithms which do this learn a mapping for the original high dimensional vector space to a lower dimensional space, either in a linear or nonlinear fashion, depending on the method, e.g. Principal Component Analysis (PCA) and t-distributed stochastic neighbor embedding (t-SNE). It is often computationally cheaper to work with lower dimensional data, and such data with lower dimension is easier to visualize as well (up to certain limit like 3). The interface implemented is as follows:

sklearn.module.DRMethod is the constructor, where you might be able to specify the target number of dimensions (depending on the algorithm).

fit behaves in the same way as with the unsupervised clustering methods, but there is no *predict* function. Instead, there is a *transform* function that transforms the inputs to the lower dimensional space after the model has been trained.

Utilities

Preprocessing

Preprocessing is a fundamental part of ML and is often the first step that is applied to the data. Operations involve scaling, centering, normalizing, among others such as feature extraction strategies for various data types such as text or image data. The majority of preprocessing functions implement the *fit* and *transform* functions in a similar fashion as the dimensionality reduction interface.

This allows you to chain various preprocessors and data reduction together to form a pipeline, where each element is a *transformer* i.e., implements *transform*, before being passed to a final estimator.

Model Selection

Model selection is the process of evaluating estimator performance, tuning hyperparameters and quantifying the quality of predictions. Broadly, this module can be divided into 3 sub modules

Metrics - The metrics module provides various methods to evaluate estimator performance on a test set for both classification and regression problems. Each method takes in two values, test_y which is the list of ground truth labels and test_pred which is the list of predictions of the estimator, and returns the score. Some metrics may require class probabilities instead of classes, such as Area under ROC curve.

Cross Validation (CV) - Scikit-learn allows you to create cross validation iterators following various strategies such as Kfold, Leave one out etc, and these iterators accept a certain metric to evaluate.

Search CV - Tuning hyperparameters is possible using a grid search or randomized search strategy. Scikit-learn supports both of them and requires you to pass in the model to be optimized, a list or range of parameter values to be searched, and a cross validation iterator set up with a particular metric, and

automatically loops over the different parameter values and gets estimates of their performance using CV with the specified metric.

1.1.6. Skimage

Scikit-Image[8] is an image processing toolbox that provides utilities for manipulating image data. Like Scikit-Learn, it is also built on top of NumPy and SciPy and offers optimized routines. Although many of the features of Scikit-Image are not necessary for ML, being able to load, store, and filter image data is necessary when dealing with images or videos. Scikit-Image also provides functions to extract various types of features from image data which can be fed as inputs to ML algorithms. Images are stored as 3D numpy arrays with the shape format of <rows, cols, channels>. Images may be of type np.uint8, having values of 0-255 per pixel, or np.float32, having float values of 0-1 per pixel.

Loading, Saving and Displaying Images

The *skimage.io* module provides these utilities

```
img = imread("image.jpg")                    # read an image as a numpy array
img = imread("image.jpg", as_grey = True)    # load image as 2D grayscale array
imshow(img)                                  # display the image
imsave("image.jpg", img)                     # save the image to file
```

Exposure Correction

The *skimage.exposure* module provides these utilities. We perform correction of exposure to remove inconsistencies in lighting conditions and to enhance certain image features

```
adjust_gamma(img)          # performs gamma correction on the image
histogram(img)             # returns histogram of the image and its color distribution
equalize_hist(img)         # uses the histogram equalization method
equalize_adapthist(img)    # performs contrast limited adaptive histogram equalization
```

Feature Extraction

The following feature extraction methods can be used to extract features from images to feed them into ML algorithms. They are present in the *skimage.feature* module

```
daisy(img)          # computes dense DAISY features, used in bag-of-features image representations
hog(img)            # computes Histogram of Oriented Gradients, used for pedestrian detection
local_binary_pattern(img)    # computes LBP features, used for fast face detection
```

Exercise 1.1.6.a. Try out the different exposure correction methods on a set of grayscale images that are bundled with skimage. Observe the differences in their results.

[8] http://scikit-image.org/

Exercise 1.1.6.b. Plot the DAISY and HOG features by visualizing them. Follow the example on the scikit-image site[9] and compare results for different images.

1.1.7. Matplotlib

Matplotlib[10] is a Python plotting library that allows you to make interactive plots. Plots can also be saved as high resolution images for publications. It simplifies the process of plotting by providing an easy to use plotting API, allowing users to write scripts to define the properties and layout of the plot. Plotting and visualization facilitates a better understanding of the data and allows users to build their ML models accordingly. Performance metrics can also be plotted to help with parameter tuning of ML models.

Sine wave[11]

```
import matplotlib.pyplot as plt
import numpy as np

t = np.arange(0.0, 2.0, 0.01)                    # generate time steps (x)
s = 1 + np.sin(2*np.pi*t)                        # generate wave (y)
plt.plot(t, s)                                   # create plot of s vs t
plt.xlabel('time (s)')                           # label x axis
plt.ylabel('voltage (mV)')                       # label y axis
plt.title('About as simple as it gets, folks')   # add title
plt.grid(True)                                   # show grid in background
plt.savefig("test.png")                          # save as png
plt.show()                                       # display interactive plot
```

Multiple plots with subplot

```
x1 = np.linspace(0.0, 5.0)                       # generate data (x)
x2 = np.linspace(0.0, 2.0)

y1 = np.cos(2 * np.pi * x1) * np.exp(-x1) # generate output (y)
y2 = np.cos(2 * np.pi * x2)

plt.subplot(2, 1, 1)                             # create a new subplot
plt.plot(x1, y1, 'o-')                           # dashed circle lines
plt.title('A tale of 2 subplots')                # title
plt.ylabel('Damped oscillation')                 # y label for subplot

plt.subplot(2, 1, 2)                             # create new subplot
plt.plot(x2, y2, '.-')                           # plot line with points as dots
plt.xlabel('time (s)')                           # add label for plot
plt.ylabel('Undamped')                           # add y label for subplot
plt.show()                                       # display plot
```

[9] http://scikit-image.org/docs/stable/auto_examples/
[10] https://matplotlib.org/index.html
[11] https://matplotlib.org/users/screenshots.html

1.1.8. t-SNE

t-Distributed Stochastic Neighbor Encoding[12](t-SNE) is a dimensionality reduction technique that is used to visualize high dimensional data. Most of the ML algorithms rely on data being fed in as multi-dimensional vectors, which are not easy to plot on a graph. However, t-SNE reduces the dimensionality down to 2 or 3 dimensions in an intelligent fashion by optimizing a loss function that focuses on preserving the local structure of the data in the lower dimensional space[13]. See a t-SNE plot for MNIST digit data in Figure 1.

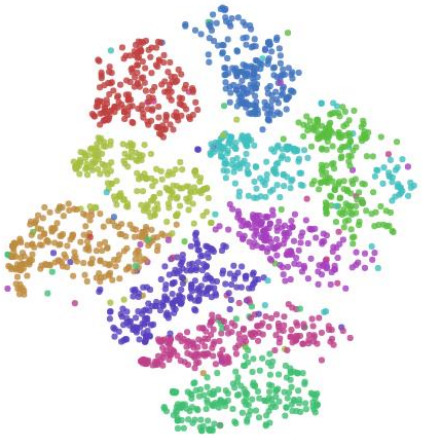


Figure 1 A t-SNE plot of MNIST data

t-SNE is implemented in Scikit-Learn as part of the Manifold Learning module with the following API

*class sklearn.manifold.***TSNE***(n_components=2, perplexity=30.0, early_exaggeration=4.0, learning_rate=1000.0, n_iter=1000, n_iter_without_progress=30, min_grad_norm=1e-07, metric='euclidean', init='random', verbose=0, random_state=None, method='barnes_hut', angle=0.5)*

Most of the parameters don't need much tuning, but in cases where the loss does not decrease, consider reducing the learning rate or the early exaggeration value. Since this is a non-convex optimization problem, different initializations will lead to different results. Simply trying a different random state may help.

t-SNE is quite computationally expensive, so you are advised to perform dimensionality reduction to a lower dimensional space of 50-100 dimensions using PCA (refer section 4.2.1), and then apply t-SNE. A GPU accelerated version of t-SNE is available outside of Scikit-Learn, which will provide a significant speedup.

Exercise 1.1.8.a. Apply t-SNE to the MNIST dataset and see if you can generate the figure above. Examine how working with a subset of the data affects the results. Optionally, try using PCA first to convert the data to 50 dimensions.

1.1.9. TensorFlow

TensorFlow is a deep learning library developed by Google with the following features:

[12] https://lvdmaaten.github.io/tsne/
[13] http://colah.github.io/posts/2014-10-Visualizing-MNIST/

- Layered Architecture with wrappers for different languages and C++ core engine for performance.
- Runs across devices – from mobiles to servers
- Portability - deploy on CPUs or multiple GPUs using single API
- Visualization tool called Tensorboard
- Auto-differentiation for backpropagation.
- Very good compile time
- Open source and very active on GitHub

Sessions:
Sessions separate definition of computation from execution. Computations in TensorFlow are made of TF operations - defined as a DAG. Once the DAG is defined, operations are ready to be executed in a session.

Hello World Example:
```
import tensorflow as tf
graph = tf.get_default_graph()
graph.get_operations() # []
input_value = tf.constant(1.0)

sess = tf.Session()  #have to create a session for evaluation
sess.run(input_value) # 1.0
```

Multiplication Example:
```
import tensorflow as tf
x = tf.constant(2.0)
y = tf.constant(3.0)
z = tf.mul(x,y)

sess = tf.Session()
out_z = sess.run(z)
```

Neural Networks Pipeline:

- Define inputs using placeholders.
- Inference - Build the graph till prediction stage.
- Loss function - Add operations required to obtain the loss in front of inference.
- Optimizer - Add to loss graph, operations to compute and apply gradients.
- Initialize the session and the variables.
- Call train step in a train loop while feeding the placeholder dictionary.
- Evaluate after certain number of train steps - use placeholder dictionary.

Softmax Example[14]:

[14] https://github.com/aymericdamien/TensorFlow-Examples/blob/master/examples/2_BasicModels/logistic_regression.py

```
x = tf.placeholder(tf.float32, [None, 784]) # mnist data image of shape 28*28=784
y = tf.placeholder(tf.float32, [None, 10]) # 0-9 digits recognition => 10 classes

# Set model weights
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))

# Construct model
pred = tf.nn.softmax(tf.matmul(x, W) + b) # Softmax

# Minimize error using cross entropy
cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(pred), reduction_indices=1))
# Gradient Descent
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

# Initializing the variables
init = tf.global_variables_initializer()

with tf.Session() as sess:
        sess.run(init)
        for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        # Run optimization op (backprop) and cost op (to get loss value)
        _, c = sess.run([optimizer, cost], feed_dict={x: batch_xs, y: batch_ys})
```

Checkpointing:

This is used to save a session state (defined by variable values) on the disk and then restore it a later point. This is useful for pretrained models and deploying models in production.

```
saver = tf.train.Saver() #object in the graph

#create session and do something

saver.save(sess, './model1.ckpt') #save the weights on the disc
saver.restore(sess, './model1.ckpt') #load weights from the disc
```

Tensorboard:

It is a powerful visualization tool that ships with TF. To use Tensorboard, add summaries for any tensor in the graph and dump the summaries to the disk periodically. The values can then be visualized on Tensorboard in the browser. Summaries can include plots across iterations, runs etc. and histograms. The data flow graph can also be visualized.

1.1.10. Keras

Keras[15] is a neural networks API that uses a separate backend such as Theano, TensorFlow or CNTK to execute the model. Keras allows users to quickly define and experiment with most of the standard neural network architectures.

Features

- Keras is a user friendly API, and allows the programmer to easily describe, compile and run models
- Keras is modular, which allows users to mix and match layers, optimizers, cost functions etc.
- Extensibility is simple with templates for writing custom layers, metrics and activation functions
- Keras can run on the CPU or GPU, depending on the backend configuration
- Keras models are portable across platforms, and can use existing tools like tensorboard

Models

Keras models come in two flavors, the Sequential model and the Functional model. The Sequential model only allows layers to be stacked linearly, whereas the Functional model allows layers to be described as arbitrary DAGs. Models need to be complied after creation with a target loss function and optimzer, along with any metrics to measure while training. Once compiled, models can be trained with the fit method where the train data, number of epochs, batch size and validation set (if any) are specified. Trained models can be easily visualized and saved to disk, for later use.

Layers

Keras layers define the operations of the neural network. There is support for all common layer types such as Dense, Convolutional and Recurrent, along with more specialized layers for advanced users. Custom layers can also be created by specifying the forward pass operation and set of learnable weights. Layer objects are callable, and can be called on tensors, and this behavior is used to specify Functional models with multi input/output layers, layers with shared weights and other complex architectures.

Losses and Metrics

A loss function is the function to be minimized during training. It is normally a measure of the error of the model. Keras provides several loss functions for both Classification and Regression tasks, and one can easily write custom loss functions with either Theano or TensorFlow, and let the backend take care of auto-differentiation. A metric is similar to a loss function and is a measure of the performance of the model, but is not used to train the model. Not all metrics can be used as loss functions.

Optimizers and Callbacks

Optimizers minimize the loss function through back propagation. Keras provides advanced optimizers and allows TensorFlow optimzers to be used as well. The learning rate of optimizers can be varied during training through Callbacks. Callbacks run at the end of each iteration and can be used for logging, monitoring, early stopping and checkpointing as well.

Examples

[15] https://keras.io/

1 Specifying a model with the Sequential API

```
from keras.models import *
from keras.layers import *

model = Sequential()
model.add(Dense(10, input_shape = (784,)))
model.add(Activation("softmax"))
```

2 Specifying models with the Functional API

```
input_tensor = Input(shape = (784,))
x = Dense(10)(input_tensor)
output_tensor = Activation("softmax")(x)

model = Model(inputs = input_tensor, outputs = output_tensor) # same model as above
```

Multi input and output model

```
inp1 = Input(shape = (50,))
inp2 = Input(shape = (100,))

x = Dense(10, activation = "relu")(inp1)
y = Dense(10, activation = "relu")(inp2)
z = concatenate([x,y])

out1 = Dense(1, activation = "softmax")(z)
out2 = Dense(1)(z)

model = Model(inputs = [inp1, inp2], outputs = [out1, out2])      # X shaped model, 2 inputs, 2 outputs
```

Compiling and training a model

```
from keras.optimizers import *

opt = Adagrad(lr = 0.0001)
model.compile(          optimizer = opt,
                        loss = "binary_crossentropy",
metrics = "accuracy")
model.fit(train_x, train_y, batch_size = 32, epochs = 10)
```

1.2. Installation Guidelines

1.2.1. Python

The Python version that you should be using is version 3.X (3.6 at the time of writing). Python 2.7 is still widely used for legacy code and libraries, but the present and future of Python is version 3.X, which has added several new features and optimizations to make your code simpler, faster and more powerful.

1.2.2. Python Libraries

The various Python libraries needed for ML, such as NumPy, SciPy and Scikit-Learn are easy to install using pip. The installation process involves the compilation of several core modules written in C and Fortran, and thus you should ensure that you have a recent version of GCC, G++ and GFortran installed. Precompiled packages are available in the form of python wheels, but they might not be optimized for your particular architecture. Also, many of the algorithms rely on BLAS (Basic Linear Algebra Subroutine) libraries to accelerate LA operations on multiple cores and will provide massive speedups in some cases. Some examples of these include Intel's Math Kernel Library (MKL) and OpenBLAS. However, installing and linking with these libraries is a somewhat involved process and not recommended for beginners.

1.2.3. Anaconda

Fortunately, Anaconda[16] is a Python distribution that is designed specifically for data science and contains hundreds of commonly used modules such as NumPy, SciPy and Scikit-Learn. Anaconda also ships with a version of MKL which these libraries are automatically linked to, so the installation process is very simple. It is also beneficial for those using compute clusters where they don't have root access, as Anaconda can be locally installed. Installing additional packages can be done the same way with pip (or pip3).

1.2.4. TensorFlow

Unlike Theano, TensorFlow has two branches, tensorflow and tensorflow-gpu, depending on which you want to use. For the GPU version, it is recommended that you install cuDNN[17] and you can gain 30-40% boost by compiling the CPU version from source by enabling SSE4 optimizations.

1.2.5. Keras

Anaconda doesn't ship with Deep Learning Frameworks such as Theano, Tensorflow, or Keras. Keras itself requires either Theano or TensorFlow to run, and sometimes one is faster than the other, so you are advised to install both. If you have access to a GPU, you can install GPU accelerated versions of these libraries to gain massive speedups. Theano can run on both AMD and Nvidia GPUs while TensorFlow is currently supported only on Nvidia GPUs. Keras configuration files allow you to change the backend by modifying the contents of ~/.keras/keras.json and Theano has a similar configuration file.

[a2]

2. Tasks

2.1. Basic Tasks

The basic tasks involved in a standard ML pipeline are loading, visualizing, and cleaning the data, followed by model fitting, evaluation, and deployment (making predictions on unseen data).

2.1.1. Data Representation

Machine Learning models typically require structured data for training and prediction (testing). Structured data is represented in a tabular form with each row representing an observation or training instance, and each column representing a particular feature of interest. Supervised Learning methods

[16] https://www.continuum.io/downloads
[17] https://developer.nvidia.com/cudnn

have an additional column for the target variable to be predicted. The Kaggle Titanic dataset[18] will be used as an example for this chapter.

Exercise 2.1.1.a. Load the dataset using Pandas, the data manipulation library that is used extensively to deal with structured data. On the resulting DataFrame, run the **head()**, **info()**, and **describe()** functions. How many rows do you see? How many columns? What are the datatypes of the features? What differences do you see between train.csv and test.csv?
Exercise 2.1.1.b. Pandas supports searching in a DataFrame similar to basic SQL queries. Use this feature to further explore the dataset. How many people survived? How many of the survivors were male? How many females below the age of 18 did not survive?

Although the dataset has been loaded into a DataFrame, many of the features need to be converted into a form that can be given as input to ML models. Each feature can broadly be classified as either a continuous value, such as age, or a categorical value, such as sex. Categorical variables need to be converted into a one hot encoded vector (or, with any other reasonable encoding strategy) before feeding them into a model. Continuous variables may need to be scaled to have zero mean and unit variance, as this is a requirement for many ML algorithms.

Exercise 2.1.1.c. Identify the categorical features in the dataset. Convert them to one hot encoded form using the **get_dummies()** function. Use the **map()** function to convert 'male' and 'female' to 0 and 1 (or, -1 and 1).
Exercise 2.1.1.d. Scale and center the continuous values using the StandardScaler class in Scikit-Learn. Verify that the data is indeed standardized by calculating the mean and variance.

Sometimes we can extract useful information from columns which we might otherwise discard. We can extract a person's title from his/her name, which may indicate their status, and this may have some correlation with the chance for survival. We can also group tickets and cabins by class.

Exercise 2.1.1.e. Extract new features from the data using the strategy mentioned above. Without training any models, can you determine if these features will be useful?

2.1.2. Data Visualisation

Visualization of features and their influence on the output helps with the process of feature engineering. One standard way of doing this is by plotting the correlation matrix for all features and the target variable which also helps to identify highly correlated features as well. Matplotlib is commonly used for visualization, but you may find it useful to try out Seaborn[19] as it provides support for common tasks like plotting feature importances etc.

Exercise 2.1.2.a. Plot the correlation matrix for the Titanic dataset, with the original features, and with the features you extracted in Exercise 2.1.1.e. You should obtain a plot[20] like the one below.

[18] https://www.kaggle.com/c/titanic/data
[19] https://seaborn.pydata.org/
[20] https://www.kaggle.com/helgejo/an-interactive-data-science-tutorial

Figure 2 A feature correlation matrix

To analyze how continuous variables affect the output, you can plot overlapping histograms, For example, if you want to see how the Age of Male passengers affect the survival rate, you can plot one histogram for all the males who survived and one for those who didn't. It looks something like the following:

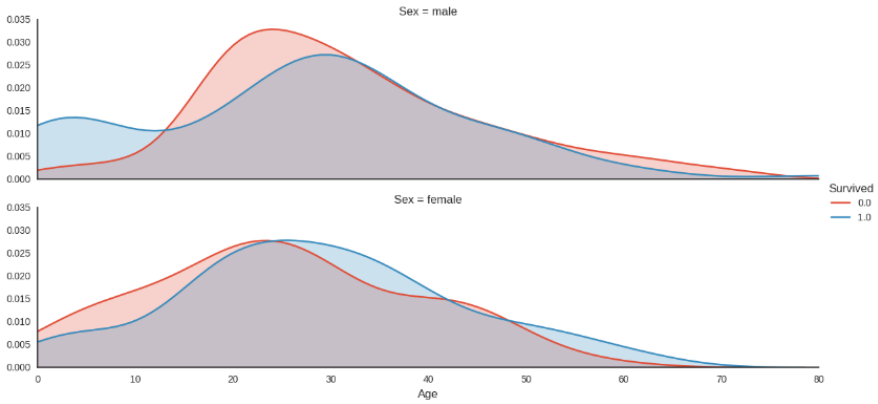


Figure 3 Histogram for survival analysis

Exercise 2.1.2.b. Make survival plots for all features like the figure above using seaborn.
Exercise 2.1.2.c. Extract Polynomial Features[21] and repeat the exercises above. Do these additional features have better correlation with the survival value?

2.1.3. Data Cleaning

Quite often, datasets have missing values due to several reasons, such as loss of records, or invalid observations. Most ML algorithms cannot automatically recognize missing values in the dataset. So, the dataset needs to be cleaned in order to feed the values into the model. Sometimes, data samples are quite imbalanced, and this causes problems while trying to learn a classification model by introducing majority class biasness. While some models viz. decision tree can deal with the imbalance automatically,

[21] http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html

others require manual rebalancing of the data. Most usual victims of majority class biasness are the mathematical models like multilayer perceptron.

Pandas allows you to easily deal with the problem of missing data by representing such values as NaNs (Not a Number) in the DataFrame. It is easy to examine which rows and columns have missing values by using the *isnull()* function.

Exercise 2.1.3.a. Explore which columns in the Titanic dataset have missing values. Print mean, median and mode of these columns. Now, for each column (feature) replace the missing value with the mean (or median) of the rest of the values of that particular column and observe the updated mean, median and mode.

Now that we've found where the missing values are, we can adopt several strategies to deal with them. You could think of two obvious approaches, though it is not usually advisable for ML. Moreover, this is again sensitivity of the problem at hand, quantity of available data etc. The first approach would be to drop rows with missing values, and the second would be to drop columns with missing values. The advantage of dropping rows over dropping columns is that you are not losing out on an entire feature (column), just because for some samples for which the features are missing. In cases where some rows have multiple missing values, this strategy is particularly useful, because each dropped column leads to a loss of information that the ML model could have used to make decisions. However, if you observe that all the missing values belong to a certain column, and there are a substantial number of missing values, dropping the column may be a good idea.

Exercise 2.1.3.b. Drop rows / columns with missing data in the Titanic Dataset using the dropna() function. How many rows do you lose, and how many columns. Which approach is better?

Sometimes, missing values may make up a small fraction of the total number of values, but might be spread out over many rows and columns. In such a case, it is not a good idea to drop rows or columns because very little data may be left over. A better approach would be to fill in these missing values as best as possible. A simple strategy is to fill in all the missing values with a constant, say 0. It is possible that the ML model may learn to account for these 0s in the data. Better approaches would be to fill in missing values with the mean, median or mode values for the column. Which one to use depends on the problem, for example using mean for placement packages is a bad idea and the median is a better estimate.

Exercise 2.1.3.c. Replace missing values with either mean median or mode. Which one should be used for categorical data? Use the *fillna()* function to do this.

Data in classification problems can be unbalanced, i.e. all classes don't have a similar number of samples. This can lead to biased classifiers, and a simple technique to deal with this is 'resampling'. If you have more samples of class A than class B, you can select as many samples from A as there are in B, to make it even. This is called downsampling (or, undersampling). Alternatively, you can select the same samples multiple times from B, to make the number of samples even. Which to use depends on the problem. There are cases in which, you may need to increase the samples of a particular class by regenerating new samples from the available ones. This process is called oversampling. Note that oversampling is more challenging and complex than undersampling.

Exercise 2.1.3.d. Analyze the Titanic dataset for class imbalance and resample the data to balance it.

2.1.4. Train, Validate, Test

The goal of an ML task is to find a model that best explains the available data. To do this, we define an ordering on models using certain metrics that are indicative of the model's predictive power. To do this, we split the dataset into a training set and a testing set. The testing set is usually a set of unseen data that helps in estimating the quality of generalization of the model. We fit the model to the training set and evaluate its performance on the test set. There shouldn't be any overlaps between these sets otherwise the model will simply memorize samples and this will lead to 'overfitting'.

The train_test_split function in *sklearn.model_selection* allows you to specify a split for a dataset, with some fraction held out as a test_set. Consider the example of Iris plant classification from the properties of its flowers.

```
iris = datasets.load_iris()
train_x, test_x, train_y, test_y = train_test_split(iris.data, iris.target, test_size = 0.3)
```

Here, test_size = 0.3 says that 30% of the dataset is kept aside for testing.

Many models require parameters to be tuned, which you might have come across in the previous section as hyperparameter tuning. Repeatedly training a model with different parameters and evaluating the performance on the test set can lead to information leaking from the test set. To avoid this, the train set is further split into a train and validation set (can be done using the same train_test_split function), and tune the parameters by observing the performance on the validation set. Once a suitable parameter configuration has been determined, the final evaluation is done on the test set.

Creation of a validation set reduces the number of samples available for training. A solution is k-fold cross validation (also called k-CV or k-fold CV), where the train set is split into k folds, with k-1 folds used for training and the remaining fold used for evaluation (validation) of the model. This is repeated k times such that evaluation is done for each of the k folds, and the metrics are averaged out. Scikit-learn provides cross validation iterators and scoring methods.

*sklearn.model_selection.***cross_val_score***(estimator, X, y=None, groups=None, scoring=None, cv=None, n_jobs=1, verbose=0, fit_params=None, pre_dispatch='2*n_jobs')*

*class sklearn.model_selection.***KFold***(n_splits=3, shuffle=False, random_state=None*)

To optimize the parameters of a model, we often search several combinations and this can be done quite effectively by setting up either a grid search that exhaustively tries all parameter combinations in a specified list, or a randomized search that randomly tries parameter combinations where each parameter is sampled from a range. The GridSearchCV and RandomizedSearchCV are the classes used for this.

*class sklearn.model_selection.***GridSearchCV***(estimator, param_grid, scoring=None, fit_params=None, n_jobs=1, iid=True, refit=True, cv=None, verbose=0, pre_dispatch='2*n_jobs', error_score='raise', return_train_score=True)*

Exercise 2.1.4.a. Write your own train_test_split function without using any loops.
Exercise 2.1.4.b. Write your own Kfold iterator and cross_val_score helper function.
Exercise 2.1.4.c. Using the cross_val_score function you wrote, implement GridSearchCV.

2.1.5. Prediction

Trained models need to be evaluated on test data (or using cross-validation) in order to estimate their predictive power. The model's performance is determined by one or more metrics that measure the model's ability to predict new data. Which metric to use depends largely on the type of problem (classification, regression, clustering etc.) and the nature of the data (balanced, unbalanced, skewed etc.)

Scikit-Learn Classifiers and Regressors have a predict function that returns the class label for classifiers and the predicted output for Regressors. Classifiers also have a *predict_proba* function that returns class probabilities for each class, with the total for all classes summing up to 1. While some metrics need only the class labels, others require the class probabilities for evaluation. All metrics are available in the *sklearn.metrics* module.

Regression Metrics

Regression metrics seek to measure how close the predicted values are to the actual values. The notion of "closeness" depends on the metric, with the Mean Absolute Error employing the L1-norm loss, and the Mean Squared Error using the L2-norm loss. Another popular metric is the R^2 (R-squared) score which measures how well a future sample is likely to be predicted.

Exercise 2.1.5.a. Implement the Mean Absolute Error (MAE), Mean Squared Error (MSE) and R^2 score using only numpy functions. Compare your results with the functions in the metrics module.

Classification Metrics

Some commonly used classification metrics are:

- Accuracy - % of correctly predicted labels
- Precision - % of predicted positive samples that are true positives
- Recall - % of true positive samples detected
- F-Score - Harmonic mean of precision and recall

All these metrics can be computed from the confusion matrix, where each element at row i, col j is the number of samples in class i, but predicted to be in class j. The name 'confusion matrix' comes from the behavioral confusion of the model while taking predictive decision on the input data.

Exercise 2.1.5.b. Write a function to compute the confusion matrix given y_true (true class output) and y_pred (model predicted class output) values. Use this confusion matrix to compute accuracy, precision, recall and F-score.

Another very useful metric is the Area Under the Receiver Operating Characteristic (ROC) curve, in short called AUC. It shows the performance of a classifier while the threshold is varied. It is created by plotting the True Positive Rate vs the False Positive rate. The AUC is 1 for a perfect classifier and 0.5 for a random

classifier, given balanced data. This function requires probability estimates to be provided to compute the area.

Exercise 2.1.5.c. Write a function to calculate the AUC given y_true and y_pred values. Verify your solution using the sklearn functions, and also plot the curve using Matplotlib.

2.2. Specialised Tasks

In many cases, data for ML is not supplied in the form of structured rows and columns, but in the form of more complex collections such as images, text, and time-series. Efficiently vectorizing such forms of data is a challenging problem but recently, Neural Networks (NN) and their variants have proven to be adept at automatically extracting features from such types of data. However, one cannot leverage their representation learning ability in all cases, and it is worthwhile to be familiar with other useful vectorizing techniques as discussed below.

2.2.1. Dealing with image data

Images can be represented as 3D arrays of pixel intensities. Feeding a flattened image to ML algorithms fails to work for moderate or large sizes, especially in image datasets with high variance. This is because spatial information is lost while flattening the image, and changes in lighting conditions can cause predictions to vary significantly. We apply exposure correction to solve the illumination variance problem, and apply local pattern preserving feature extractors to vectorize images.

Dense DAISY Features and visual bag of words model

DAISY features are gradient orientation based features extracted from local patches of an image. These features have the advantage of being robust to image transformations. However, these are dense features and can cause the dimensionality of the data to increase significantly if computed for each patch of pixels. This defeats the purpose of using an image feature extractor.

DAISY Features are extracted only for certain automatically determined key points in an image (a region characterized by change in texture, such as an edge or a corner). The extracted features from all images are then clustered together into K clusters, and each cluster represents a “visual word”, forming a vocabulary of size K. Then, a histogram is constructed for each image, to count the frequency of the K “visual words” in an image, and the normalized K dimensional vector can be used as a feature vector.

Histogram of Oriented Gradients and Local Binary Patterns

HOG features are again based on image gradients. Specifically, the image is divided into blocks, with each block consisting of a group of cells. The average gradient orientation is computed for each cell, and a histogram is constructed by binning the orientations. A vector consisting of these normalized descriptors for each block is used for ML. HOG features work best to detect objects with distinct edges, such as pedestrians, cars etc.

Local Binary Patterns form a dense descriptor by comparing the intensity of a pixel to its neighboring pixels thus creating a binary vector. These patterns are good at modeling texture, and have been applied to fast face detectors due to their computational efficiency.

Exercise 2.2.1.a. Read the paper at [1] on Scene Recognition and try to implement the feature extraction method used in it. How does it compare with the authors' implementation shared at [2]?

[1] https://arxiv.org/pdf/1702.06850.pdf (*accessed May 2017*)

[2] https://github.com/flytxtds/scene-recognition/blob/master/scene_recognition_local_global_approach.ipynb

2.2.2. Dealing with text data

Converting text data to a fixed dimensional vector form is a difficult problem due to several factors such as variable length sentences and a large vocabulary of words. Sentences can be padded with null characters to make them the same length, but one hot encoding words as vectors will make the representation extremely sparse and difficult to work with.

Bag of Words

The Bag of Words approach converts each token (word or character) to a unique integer and then creates an N-dimensional histogram of the word frequencies (uni-grams) in the document, where N is the size of the vocabulary. This approach loses information about connectivity between words. To overcome this, histograms can be made for each pair of consecutive tokens, to form bi-grams, at the cost of more space.

Scikit-Learn API

*class sklearn.feature_extraction.text.***CountVectorizer***(input=u'content', encoding=u'utf-8', decode_error=u'strict', strip_accents=None, lowercase=True, preprocessor=None, tokenizer=None, stop_words=None, token_pattern=u'(?u)\b\w\w+\b', ngram_range=(1, 1), analyzer=u'word', max_df=1.0, min_df=1, max_features=None, vocabulary=None, binary=False, dtype=<type 'numpy.int64'>)*

Exercise 2.2.2.a. Load the 20 Newsgroups dataset[22] and vectorize it using uni-grams and bi-grams. What difference do you see in the size of the vector representation? What about its sparsity?

TF-IDF Vectorizer

Some words, such as "a", "an" and "the", occur very commonly in text passages but do not contribute much in terms of the content of the passage. Hence, it makes sense to assign these frequencies a low weight, so that other, more meaningful words are given higher importance. The TF-IDF vectorizer calculates the Term frequency (TF) for a word in a passage, and multiplies that with the Inverse Document Frequency (IDF), which is a measure of how uncommon a word is.

Scikit-Learn API

[22] http://qwone.com/~jason/20Newsgroups/

*class sklearn.feature_extraction.text.***TfidfVectorizer***(input=u'content', encoding=u'utf-8', decode_error=u'strict', strip_accents=None, lowercase=True, preprocessor=None, tokenizer=None, analyzer=u'word', stop_words=None, token_pattern=u'(?u)\b\w\w+\b', ngram_range=(1, 1), max_df=1.0, min_df=1, max_features=None, vocabulary=None, binary=False, dtype=<type 'numpy.int64'>, norm=u'l2', use_idf=True, smooth_idf=True, sublinear_tf=False)*

Exercise 2.2.2.b. Use the TF-IDF vectorizer on the 20 Newsgroups dataset and report which words have the lowest and highest IDF. Are they what you would expect?

2.2.3. Dealing with time-series data

Time series data consists of a sequence of events occurring at successive time steps. Like text data, it is challenging to model because a meaningful representation has to take into account a variable number of events that occurred in the past, and assign importance to each of them. Although this is a task that is particularly well suited for a class of neural networks called Recurrent Neural Networks (RNN), there are other, more basic techniques that can be applied as well. Note that, standard statistical models are also quite powerful in dealing with time-series data.

Forecasting with a Sliding Window

Given the task of predicting future events based on past data, a simple technique is to use a fixed number of time steps (a window) and use that as a feature vector to predict the next time step. Formally, we wish to learn a function f such that $E_t = f(E_{t-1}, E_{t-2}, \ldots E_{t-n})$ where E is the event at time t, and n is the size of the window. One can also weight each event based on its importance, or the time of occurrence and use this to attempt to improve prediction quality.

Exercise 2.2.3.a. Implement the sliding window technique as a class to vectorize a given time-series. It should have a transformation method to do this, and the constructor should have an optional weights parameter.

Dynamic Time Warping

Dynamic time warping (DTW) is a powerful technique to compute the similarity between two time series sequences. It is used to align them together, and in the process computes a score of how well aligned, or similar they are. This technique is popularly used with spatial clustering and classification algorithms to categorize time series data. It has been applied to EEG data to predict seizures and also for Human Activity Recognition using accelerometer data, and gesture recognition, as shown in the figure[23].

[23] https://github.com/markdregan/K-Nearest-Neighbors-with-Dynamic-Time-Warping

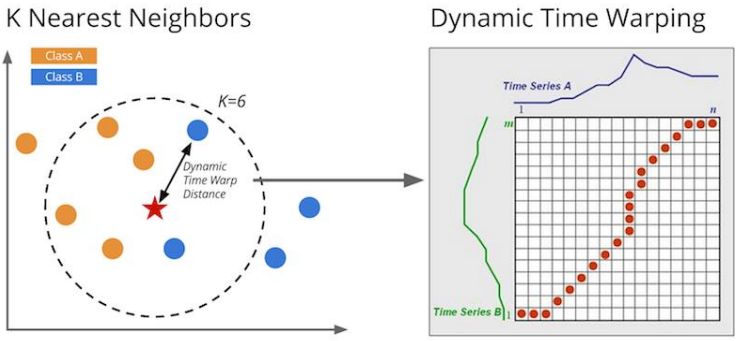


Figure 4 K Nearest Neighbor (KNN) with DTW

Exercise 2.2.3.b. Analyze the strategy for Human Activity Recognition used at [1]. Try to speed it up by using the Fast-DTW[24] algorithm available on PyPI[25]. Measure the performance improvements.

[1] https://github.com/markdregan/K-Nearest-Neighbors-with-Dynamic-Time-Warping

3. Simple Techniques

3.1. Regression

Regression techniques are used in cases when the target variable (prediction value) for the model has a continuous domain.

3.1.1. Linear regression

We start with the problem of predicting the 100m race winning time based on the Olympics year – given some supervised data as in the table below. Visualizing the data can give insights about what model to use.[26]

[24] http://cs.fit.edu/~pkc/papers/tdm04.pdf
[25] https://github.com/slaypni/fastdtw
[26] Rogers, Simon and Girolami, Mark. *A First Course in Machine Learning*. Boca Raton: CRC Press, 2012.

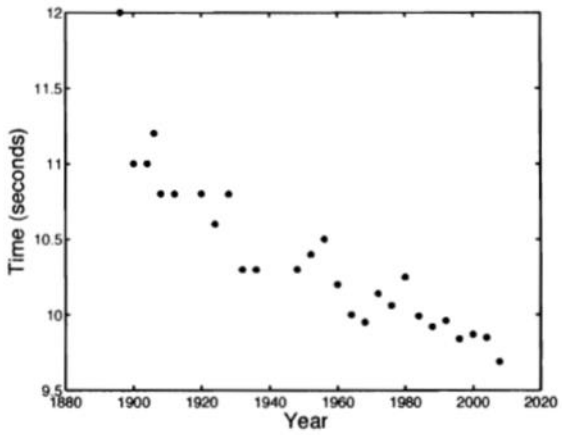
Men's Race Year-wise

x_n	t_n
1896	12.00
1900	11.00
1904	11.00
1906	11.20
1908	10.80
1912	10.80
1920	10.80
1924	10.60
1928	10.80
1932	10.30
1936	10.30
1948	10.30
1952	10.40
1956	10.50
1960	10.20
1964	10.00
1968	9.95
1972	10.14
1976	10.06
1980	10.25
1984	9.99
1988	9.92
1992	9.96
1996	9.84
2000	9.87
2004	9.85
2008	9.69

Women's Race Year-wise

x_n	t_n
1928	12.20
1932	11.90
1936	11.50
1948	11.90
1952	11.50
1956	11.50
1960	11.00
1964	11.40
1968	11.00
1972	11.07
1976	11.08
1980	11.06
1984	10.97
1988	10.54
1992	10.82
1996	10.94
2000	11.12
2004	10.93
2008	10.78

Men's Race data plot



Women's Race data plot

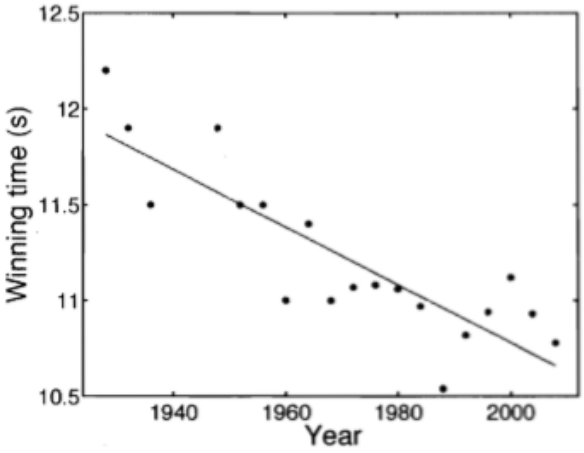


Figure 5 Olympic Data

.

From the visualization, a linear function looks like a good candidate to represent the data.

Scikit-learn API:

Sklearn offers the class LinearRegression as follows:
class sklearn.linear_model.**LinearRegression**(*fit_intercept=True*, *normalize=False*,*copy_X=True*,*n_jobs=1*)

Exercise 3.1.1.a. Refer to the sklearn documentation and implement the code to get the weights for the linear model for men's race data and plot the line using matplotlib.
Exercise 3.1.1.b. After obtaining the line, make predictions using the predict function of the learnt model for the year 2012 and 2016 for men's race. How close are the predictions to the real values?
Exercise 3.1.1.c. Use the learnt model to predict the winning times for women's race in 2012 and 2016.

Exercise 3.1.1.d. Can we learn a linear model by adding another feature for the Gender which takes the value 0 for Men and 1 for Women times? How accurate are the predictions for 2012 and 2016 for both categories?

Algorithm Implementation Notes using Gradient Descent:

- Regression parameters can be computed using the Maximum Likelihood Estimate (MLE) or using gradient descent.
- All operations can be implemented as vector operations using Numpy. Avoid for loops.
- Instead of updating for each example (online training), or for the entire dataset at once (batch training), a mini-batch can be used. This leads to faster convergence and saves compute.
- Centering of data points can be helpful to prevent values from becoming NaN.
- Decaying learning rate with time also leads to better convergence. This is usually done by setting up a decay routine (learning rate is function of iteration number).
- **Bias Trick:** The equation Y = **w**.**X** + b can be written as a dot product by combining b with **w** and appending the constant 1 as the first feature for all data points. Y = **w**.**X** + b.1 = **w'**.**X'**

Code [27] for Linear Regression in Numpy:

```
import numpy as np
class LinearReg:
        def __init__(self):
                self.weights = np.random.randn(2)              # initialize weights in the constructor

        def gradient(self,x, y):
                y_estimate = x.dot(self.weights).flatten()     # predict and convert to 1D
                error = (y.flatten() - y_estimate)             # calculate error
                gradient = -(1.0/len(x)) * error.dot(x)        # compute gradients
                return gradient, error**2

        def predict(self,X):
                return np.dot(X,np.transpose(self.weights))

        def train(self,X,Y):
                for i in range(no_of_iterations):              # gradient descent loop
                        grad,error = self.gradient(X,Y)
                        self.weights = self.weights – learning_rate*grad
                        print "Error at iteration ", i, ": ",np.sum(error)/len(X)
```

Exercise 3.1.1.e. Try to find the optimal learning rate using the Olympics data and plot the loss for each setting with respect to the iteration number. Data centering might be required to obtain convergence.

Regularization term can be added to keep the weights for the model small and prevent overfitting.

Exercise 3.1.1.f. Add the L2 regularization term to the cost function in the above implementation and find the optimal value of regularization coefficient using the Olympics data.

3.1.2. Perceptron

The idea of linear regression can be extended to binary classification problems - by drawing a hyperplane separating the two classes. The output of linear regression (v) is followed by a step function:

$$y = \begin{cases} 0 & v < 0 \\ 1 & v >= 0 \end{cases}$$

[27] Implementation from https://www.cs.toronto.edu/~frossard/post/linear_regression/

The learning rule is given by:

$$w_i(n+1) = w_i(n) + \eta[d_i - y_i]x_i$$

Considering the problem of the Boolean gate AND – we want to draw a decision boundary that correctly classifies the output of the gate[28].

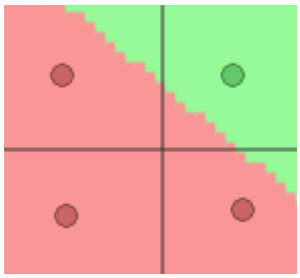


Figure 6 Visualisation of decision boundary for the output of the AND gate (function)

Exercise 3.1.2.a. Considering the following class structure and the implementation of linear regression, implement the class for perceptron in Numpy.

```
class Perceptron:
        def __init__(self):
                self.weights = np.random.rand(1,3)                    # initialize weights

        def train(self,X,Y):
                # training loop

        def predict(self,X):
                # return prediction for a data point
```

Exercise 3.1.2.b. Check if the implementation converges for AND and OR gate and plot the decision boundary.
Exercise 3.1.2.c. Does the implementation converge for XOR gate? Why/Why not? Can you visualize the decision boundary for XOR gate with two inputs?

Scikit-Learn API:

class sklearn.linear_model.**Perceptron**(*penalty=None*, *alpha=0.0001*, *fit_intercept=True*, *n_iter=5*, *shuffle =True*, *verbose=0*, *eta0=1.0*, *n_jobs=1*, *random_state=0*, *class_weight=None*, *warm_start=False*)

Exercise 3.1.2.d. Refer to the sklearn documentation of Perceptron and obtain the weights for visualization for the Boolean OR gate problem.

3.2. Probabilistic modelling

3.2.1. Logistic regression

[28] Visualization generated from http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

Consider the following dataset[29] which shows number of hours each student spent studying and whether they passed the exam.

Hours	0.50	0.75	1.00	1.25	1.50	1.75	1.75	2.00	2.25	2.50	2.75	3.00	3.25	3.50	4.00	4.25	4.50	4.75	5.00	5.50
Pass	0	0	0	0	0	0	1	0	1	0	1	0	1	0	1	1	1	1	1	1

Figure 7 Student dataset

Visualizing the data gives us the following:

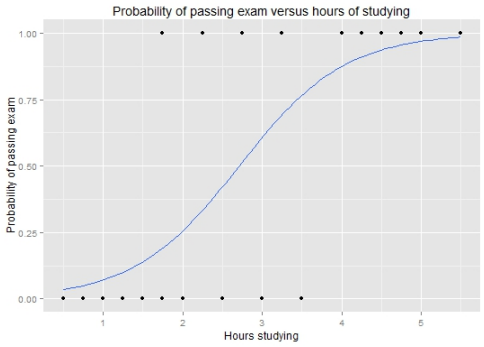


Figure 8 Visualisation of student data given in Figure 7

The goal is to learn the sigmoid function as shown in the plot – predicting the probability that the student will pass the exam given the number of hours they spent studying.

Scikit-Learn API:

class sklearn.linear_model.**LogisticRegression**(*penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='liblinear', max_iter=100, multi_class='ovr', verbose=0, warm_start=False, n_jobs=1*)

Exercise 3.2.1.a. Refer to the sklearn documentation and implement the code to get the parameters for the model for the given data and plot the sigmoid using matplotlib.
Exercise 3.2.1.b. Predict the probability for a student passing the exam who studied for 2.8 hours.
Exercise 3.2.1.c. Given the class structure below, implement the code for logistic regression in Numpy.

```
class LogisticReg:
        def __init__(self):
                # initialize weights

        def train(self,X,Y):
                # training loop

        def predict(self,X):
                # return prediction for a data point
```

3.2.2. Naïve Bayes

Consider the following data that was collected during Wimbledon indicating if the match was played on a given day subject to weather conditions.

[29] Dataset and visualization from https://en.wikipedia.org/wiki/Logistic_regression

Day	Outlook	Temperature	Humidity	Wind	Play
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Figure 9 Wimbledon play data

Exercise 3.2.2.a. Calculate using Numpy P(Y=No) and P(Y=Yes) for the given data.
Exercise 3.2.2.b. Extend the program in the previous exercise to calculate the probability of the match being played given the forecast vector of a particular day.
Exercise 3.2.2.c. Assuming Day 15 has the following forecast:

<Outlook = Sunny, Temperature = Cool, Humidity = High, Wind = Strong>

Calculate the probability of the match being played on Day 15, given the forecast.

Exercise 3.2.2.d. Is the probability obtained for the match being played given the forecast, a good estimate? What could be the reason?

Scikit-learn API:

class sklearn.naive_bayes.**GaussianNB**(*priors=None*)

Exercise 3.2.2.e. Refer to the sklearn documentation for Naïve Bayes and obtain the probability estimates for Day 15 given the forecast.

Now that we have Naïve Bayes implemented and working on a small dataset – let us consider a real world problem where Naïve Bayes really shines. This problem deals with classification of SMS data as spam or ham. The dataset consists of raw SMS data labelled as Spam or Ham.

The dataset can be found on Kaggle - https://www.kaggle.com/uciml/sms-spam-collection-dataset

Exercise 3.2.2.f. Process the data using techniques discussed in dealing with text data section and obtain a feature representation for each SMS.
Exercise 3.2.2.g. Using sklearn, build a Naïve Bayes classifier to classify SMS has spam or ham and report the accuracy.

3.3. Classification

3.3.1. Trees

We now wish to explore a white box model – given a situation, how the model made the decision is human interpretable. Modeling the classification task using a decision tree provides a good mechanism for doing so.

Consider the Iris flower dataset - available at https://en.wikipedia.org/wiki/Iris_flower_data_set. The dataset can also be obtained using scikit-learn from **sklearn.datasets**. The length and width of the sepals and petals for a flower is known and the data points are from three species. We want to build a classification model for categorizing these flowers into the three species.

Scikit-learn API:

class sklearn.tree.**DecisionTreeClassifier**(*criterion='gini'*, *splitter='best'*, *max_depth=None*, *min_samples_split=2*, *min_samples_leaf=1*, *min_weight_fraction_leaf=0.0*, *max_features=None*, *random_state=None*, *max_leaf_nodes=None*, *min_impurity_split=1e-07*, *class_weight=None*, *presort=False*)

Exercise 3.3.1.a. Refer to the sklearn documentation and build a decision tree classifier using the default parameters.
Exercise 3.3.1.b. Visualize the tree learnt in previous exercise.
Let us visualize the decision surface for the tree, using two features at a time. Hence, obtaining six plots[30] as shown below.

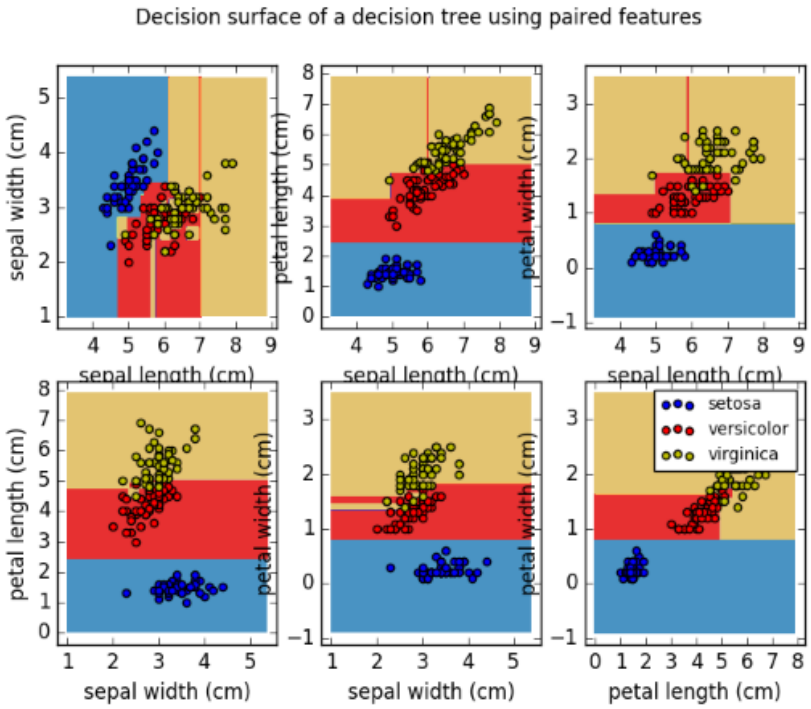


Figure 10 Decision boundary for Iris dataset (2 features at a time)

Exercise 3.3.1.c. Compute the class probabilities for each data point in the validation data. Convert the probabilities to hard label assignments and obtain the accuracy for the model.
Exercise 3.3.1.d. Instead of Gini index, use entropy for the information gain and compute the accuracy.

[30] Visualization from http://scikit-learn.org/stable/modules/tree.html

Exercise 3.3.1.e. Instead of making the best split, make the best random split and visualize the new tree model to observe the changes.

3.3.2. Rules

Extending the examples of decision trees, we are now interested in building and learning rules for classification for the Iris dataset.

Exercise 3.3.2.a. For each point in the validation set, obtain the decision path taken by the tree learnt.
Exercise 3.3.2.b. Obtain the rules for each class using the decision tree.

The rules obtained from a decision tree can be complex. Hence, we shall try to learn rules by modeling learning as search starting with the simplest rule (most general).

Exercise 3.3.2.c. Establish the baseline for accuracy by using ZeroR on Iris.
Exercise 3.3.2.d. Compute the accuracy for OneR by choosing relevant thresholds for each feature.
Exercise 3.3.2.e. Define the generalization and specialization operations and perform the search to find the rule that works the best on the given data.

3.4. Clustering

3.4.1. K-Nearest Neighbor (kNN)

Consider the problem of MNIST digits – 28*28 dimensional greyscale image for a single handwritten digit from 0-9. The complete dataset can be obtained from http://yann.lecun.com/exdb/mnist/

Since the data is high dimensional (784 dimensions), we visualize the training set using t-SNE which gives the following plot [31]:

Figure 11 t-SNE visualisation of MNIST data

This suggests that the kNN classifier might do a good job at classifying the digits.

[31] Visualization from http://colah.github.io/posts/2014-10-Visualizing-MNIST/

Scikit-learn API:

class sklearn.neighbors.**KNeighborsClassifier**(*n_neighbors=5*, *weights='uniform'*, *algorithm='auto'*, *leaf_size=30*, *p=2*, *metric='minkowski'*, *metric_params=None*, *n_jobs=1*, ***kwargs*)

Exercise 3.4.1.a. Refer to the sklearn documentation and classify the points in the validation data with default parameters using accuracy as the performance metric.
Exercise 3.4.1.b. Change the distance metric from Euclidean to Manhattan distance and compare them using the accuracy obtained on the validation data.
Exercise 3.4.1.c. Optimize the value of **k** using the accuracy on the validation data for both distance metrics.
Exercise 3.4.1.d. Is the same value of **k** optimal for both distance metrics? Reason.

Exercise 3.4.1.e. Implement the code for kNN using Numpy given the following class structure:

```
class kNN:
    def train(self,X,Y):
        # load dataset
    def predict(self,X):
        # return prediction for a data point
```

Exercise 3.4.1.f. Optimize the k-nearest search in the above implementation using K-dimensional trees. (**sklearn.neighbors.KDTree)**

3.4.2. k-means

Suppose the MNIST dataset in the previous section was unsupervised - the labels for the images in the training data were only available for evaluation and not for training. The t-SNE visualization in the previous section indicates that k-means clustering might give a similar result.

Scikit-learn API:

class sklearn.cluster.**KMeans**(*n_clusters=8*,*init='k-means++'*, *n_init=10* ,*max_iter=300*, *tol=0.0001*, *precompute_distances='auto'*, *verbose=0*, *random_state=None*, *copy_x=True*,*n_jobs=1*,*algorithm='auto'*)

Exercise 3.4.2.a. Refer to the sklearn documentation and cluster the digits of MNIST into 10 clusters.
Exercise 3.4.2.b. Calculate the Homogeneity, Completeness, V-measure and Silhouette Coefficient for the clusters. Refer - http://scikit-learn.org/stable/modules/clustering.html#clustering-evaluation
Exercise 3.4.2.c. Plot the V-measure and the Silhouette Coefficient as a function of number of clusters.
Exercise 3.4.2.d. Plot the V-measure and the Silhouette Coefficient as a function of number of iterations.
Exercise 3.4.2.e. Implement k-means in Numpy using the following class structure:

```
class kmeans:
    def __init__(self, k, X):
        # initialize k centroids
    def assign(self):
        # assign centroids to each point
    def update(self):
        # computes and returns updated centroids
    def cluster(self):
        # returns cluster number of each data point, with the centroids
```

Cases when k-means doesn't work[32]

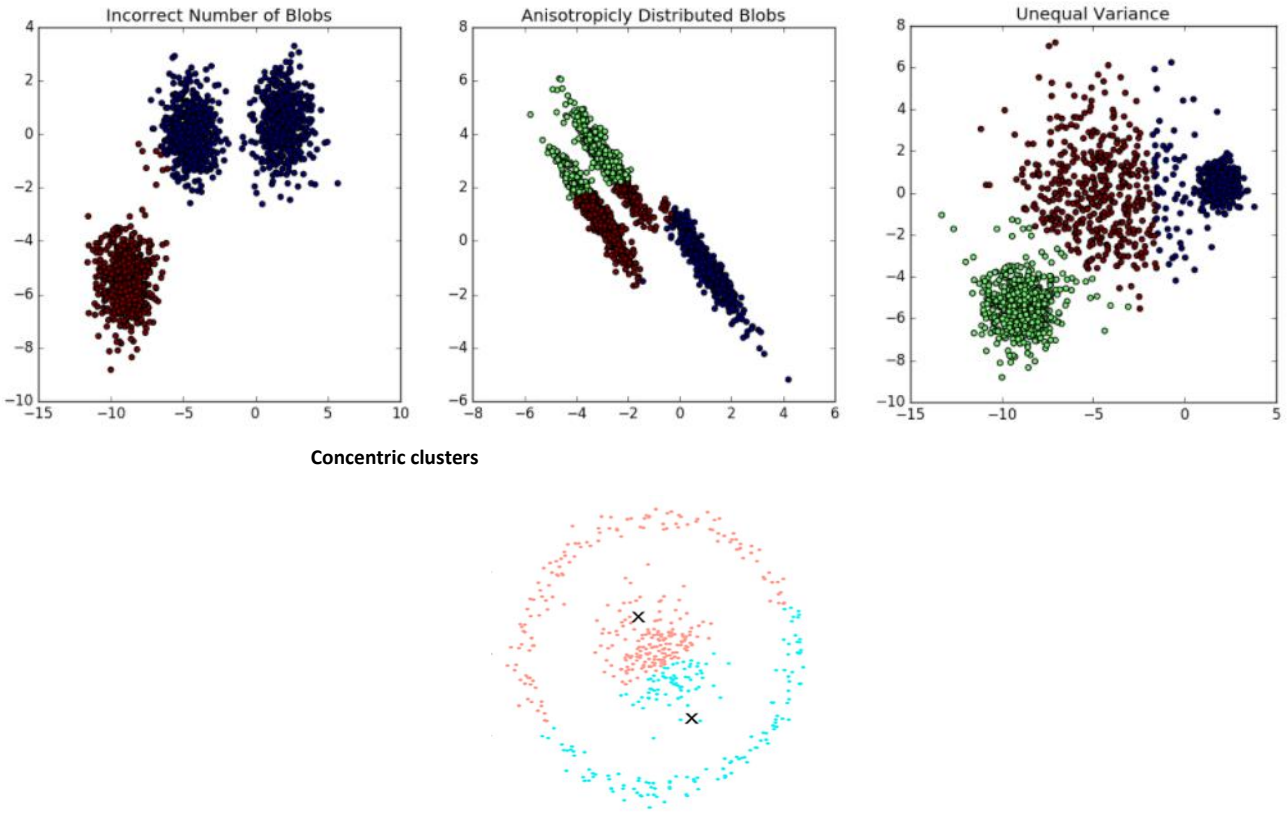


Figure 12 Datasets of 2-dimensional points in which k-means does not work efficiently

4. Advanced Techniques

4.1. Supervised Learning

4.1.1. Regression Trees

Previously, we saw how trees can be used for classification. They can also be used for regression by learning a set of rules to fit a curve in steps. The training procedure is a greedy algorithm where the optimal split is calculated at each node to minimize the error.

Scikit-Learn API

*class sklearn.tree.***DecisionTreeRegressor***(criterion='mse', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_split=1e-07, presort=False)*

Exercise 4.1.1.a. Generate a sine wave with 1,000 points and add some noise to the y values. Now fit decision trees with different values of max_depth and plot the outputs along with the original points as shown in the figure[33] below. At what point does the tree start overfitting?

[32] Image from http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_assumptions.html#sphx-glr-auto-examples-cluster-plot-kmeans-assumptions-py

[33] http://scikit-learn.org/stable/auto_examples/tree/plot_tree_regression.html

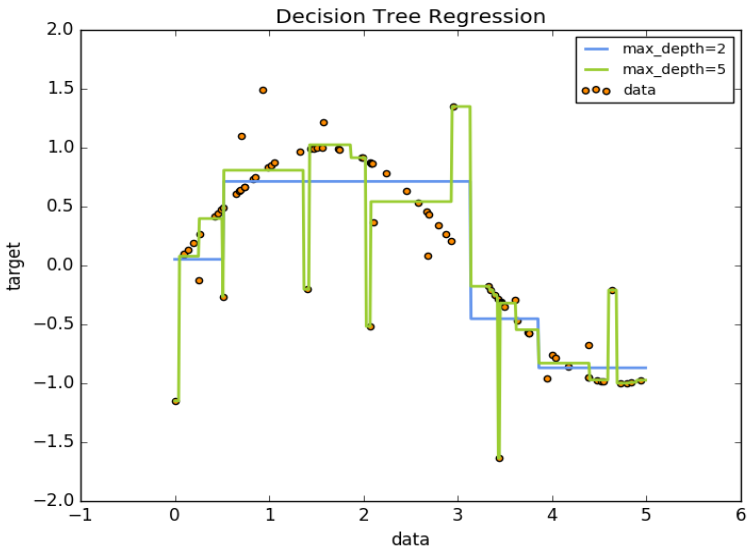


Figure 13 Plot obtained from Regression Tree

Exercise 4.1.1.b. Repeat the exercise above by making 10 trees, each trained on a random half of the data and average their predictions to get a single answer. Does this ensemble perform better than the individual trees? Do more trees yield better results? At what points do you see diminishing returns?

4.1.2. XGBoost

XGBoost is a powerful supervised machine learning algorithm that is short for Extreme Gradient Boosting. As the name suggests, it is a member of the family of boosting algorithms and is an optimized variant of the Gradient Boosting algorithm. These optimizations include methods to control overfitting through the use of regularization parameters to control model complexity, and improve predictive accuracy with dropout, a technique popularly used in neural networks which has been also been modified for gradient boosting trees. The figure[34] below shows how XGBoost attempts a tradeoff between model performance *L(f)* and model complexity *Ω(f)* while fitting the individual decision tree regressors in the ensemble.

[34] http://xgboost.readthedocs.io/en/latest//

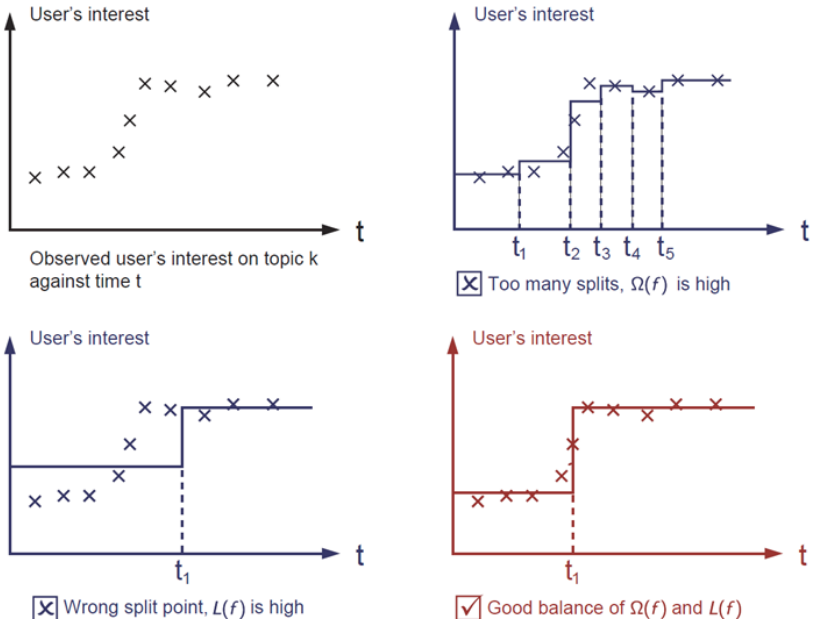


Figure 14 Trade-off between model performance and model complexity in XGBoost

XGBoost API (Note - XGBoost is a standalone library and is not part of Scikit-Learn)

class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True, objective='reg:linear', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child_weight=1, max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_lambda=1, scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

Exercise 4.1.2.a. Train an XGBoost classifier on the Kaggle titanic dataset (Section 2.1) and use cross validation to observe the tradeoff between the learning rate[35] and number of estimators.
Exercise 4.1.2.b. Plot the feature importance of the model you have trained using the xgboost plotting API and list the most important features.

4.1.3. Support Vector Machines

Consider the 20 News Groups dataset, which can be found at the following link: http://qwone.com/~jason/20Newsgroups/. The dataset contains news articles from 20 different topics and the task is to classify articles into these categories.

The first task is to obtain the feature representation for each article which is then followed by a classifier which would try to learn the decision boundary– in this case we shall look at how a support vector machine would perform.

Exercise 4.1.3.a. Obtain the TF-IDF feature representation for all documents using the sklearn inbuit functionality.

[35] http://xgboost.readthedocs.io/en/latest//parameter.html

Scikit-learn API:

Sklearn offers three tools for SVM – LinearSVC, SVC (with non-linear kernel) and SVR (Support Vector Regression). A non-linear kernel doesn't scale well for a large number of examples as we shall see in the following exercises. We need classification for this task and the two APIs are as follows:

class sklearn.svm.**LinearSVC**(*penalty='l2', loss='squared_hinge', dual=True, tol=0.0001, C=1.0, multi_class='ovr', fit_intercept=True, intercept_scaling=1, class_weight=None, verbose=0, random_state=None, max_iter=1000*)

class sklearn.svm.**SVC**(*C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape=None, random_state=None*)

Exercise 4.1.3.b. Using LinearSVC, compute the validation accuracy while tuning the hyperparameters.
Exercise 4.1.3.c. Use SVC with rbf kernel to obtain the validation accuracy.
Exercise 4.1.3.d. Try using polynomial kernels of varying degrees for the SVM model.
Exercise 4.1.3.e. Obtain the support vectors for each of the kernels.
Exercise 4.1.3.f. Compare the training time for each of the kernels.

4.1.4. Random Forest

A Random Forest is an ensemble of decision trees that fits a lot of decision trees on various sub-samples of the data, preventing overfitting and improving performance (decision accuracy) by averaging over the results of the trees.

Consider the problem discussed in the Decision Tree section. We shall now see how a random forest outperforms decision trees for the same problem.

Scikit-learn API:
class sklearn.ensemble.**RandomForestClassifier**(*n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None,min_impurity_split=1e-07, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False, class_weight=None*)

Exercise 4.1.4.a. Use entropy based and Gini index as the information gain metrics and obtain the accuracy on the validation data. Compare the accuracy obtained to decision tree model.
Exercise 4.1.4.b. Plot the accuracy with respect to the number of trees used in the ensemble. Note the diminishing returns.
Exercise 4.1.4.c. Change the number of features used to obtain a split and the plot the accuracy.

4.2. Unsupervised Learning

4.2.1. Distribution Estimation: Bayesian Networks

Bayesian networks are graphical models that represent random variables and their conditional dependencies through a directed acyclic graph (DAG). They can be used in an unsupervised fashion to learn the distribution of variables and then infer the values of hidden variables given observed values for another set of values.

A good implementation of Bayesian Networks is contained in the package "pomegranate"[36] which can be installed via pip.

Bayesian Network API

class pomegranate.BayesianNetwork.BayesianNetwork(states, graph)

*Exercise 4.2.1.a.*Consider the Wimbledon weather dataset used for the Naive Bayes exercise. Fit a Bayes Net model to this data and plot the relationships among variables using the plotting functionality. Refer to the documentation[37] and tutorials[38],[39] for the plotting functions.

[36] http://pomegranate.readthedocs.io/en/latest/
[37] http://pomegranate.readthedocs.io/en/latest/BayesianNetwork.html
[38] https://github.com/jmschrei/pomegranate/blob/master/tutorials/Tutorial_4_Bayesian_Networks.ipynb
[39] https://github.com/jmschrei/pomegranate/blob/master/tutorials/Tutorial_4b_Bayesian_Network_Structure_Learning.ipynb

4.2.2. Dimensionality Reduction: PCA

Principal Component Analysis (PCA) is an unsupervised learning algorithm that decomposes a multi-variate distribution into a set of orthogonal components that correspond to the maximum amount of variance. By discarding the components with the least variance, PCA can be used for dimensionality reduction. The figure[40] below shows how PCA has been used to reduce the dimensionality of the Iris dataset from 4 dimensions to two dimensions.

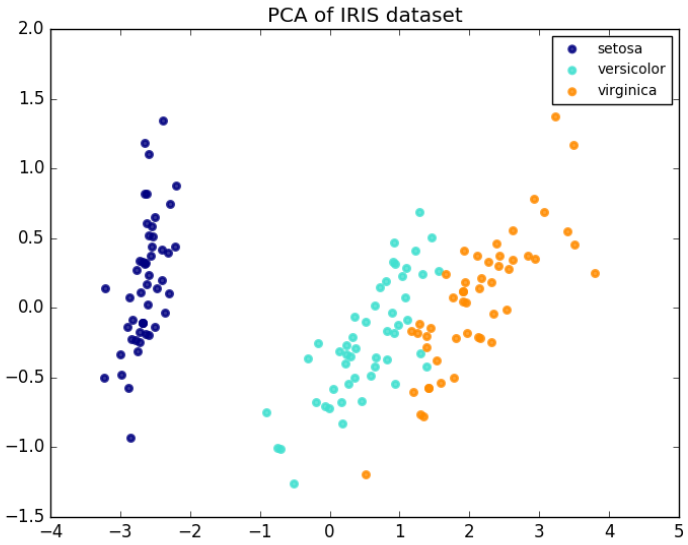


Figure 15 PCA on Iris dataset

Scikit-Learn API

*class sklearn.decomposition.**PCA**(n_components=None, copy=True, whiten=False, svd_solver='auto', tol=0.0, iterated_power='auto', random_state=None)*

Exercise 4.2.2.a. Perform PCA on the Iris dataset and generate the figure above. Perform PCA again and reduce the dataset to a single dimension. What percentage of the variance cannot be explained with just a single dimension?

Exercise 4.2.2.b. Perform PCA on the MNIST dataset and reduce it to two dimensions. Plot the results and compare it with t-SNE. Again, evaluate what percentage of the variance cannot be explained with just 2 dimensions

Exercise 4.2.2.c. How many dimensions are needed to explain 90% of the variance of the MNIST dataset? Plot the number of features versus the percentage of variance explained on the MNIST dataset and comment on the trend.

[40] http://scikit-learn.org/stable/auto_examples/decomposition/plot_pca_vs_lda.html

4.3. Deep Learning

4.3.1. Multi-Layer Perceptron

We looked at the XOR problem while dealing with Perceptron and visualized that learning the XOR function is not linear.[41]

Perceptron on XOR

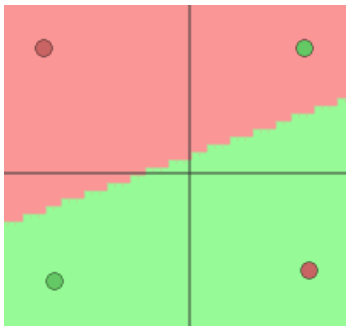


Figure 16 A bad decision boundary obtained by Perceptron on XOR dataset

Scikit-learn API:

*class sklearn.neural_network.**MLPClassifier**(hidden_layer_sizes=(100,), activation='relu', solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08)*

Exercise 4.3.1.a. Add a hidden layer with sigmoid activation and 2 hidden neurons and obtain the following visualization.

Two hidden neurons

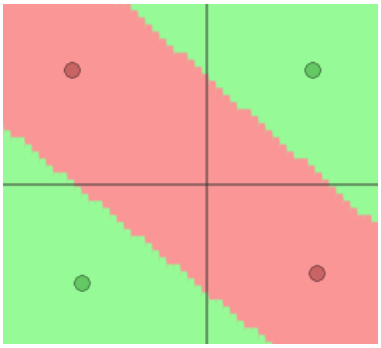


Figure 17 Decision boundary obtained by 3 perceptrons stacked as (2,1) forming a hidden layer between them

Exercise 4.3.1.b. Increase the number of neurons to 6 in the hidden layer to visualize the function learnt.

[41] Visualization from http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

Exercise 4.3.1.c. Change the activation function to **tanh** and check the speed of convergence.
Exercise 4.3.1.d. Change the activation function to **ReLU**. Is there anything special about the shape of the learnt function?

XOR problem is trivial. Let us move to a more complex problem of classifying MNIST digits – the one we had seen in the section for kNN.

A fully connected neural network, called Multilayer Perceptron (MLP) will have a bunch of hidden layers with each neuron connected to all neurons in the previous layer and the output layer having the softmax activation to obtain class probabilities in case of classification tasks. The loss can then be calculated and the training occurs using error back propagation algorithm that uses gradient descent (or stochastic gradient descent). After obtaining the gradients, the weight update step is made using an optimization rule, most common ones being SGD, Momentum, Nesterov Momentum, Adagrad, RMSProp and Adam.

Exercise 4.3.1.e. Set up a basic Fully Connected Neural Network in TensorFlow with the final layer being Softmax and the loss function as cross-entropy. Use the activation function as ReLU with standard normal weight initialization. Use Adam as the optimizer.
Exercise 4.3.1.f. What is the value of the loss for the first forward pass of a mini-batch of data? It should be approximately **–ln(0.1)** if everything is correctly setup and the weights are initialized properly.
Exercise 4.3.1.g. Report the accuracy of the model on the validation set.
Exercise 4.3.1.h. Try changing the number of hidden layers and the number of neurons in each layer. What happens to the accuracy of the model with a deeper network?
Exercise 4.3.1.i. Add L2 regularization to the loss function and plot the loss with respect to the number of iterations using TensorBoard.

Dropout is another technique for regularization. In dropout, a neuron is active with a certain probability during training. To maintain the same expected value of the output during test, the output of the neuron is scaled by the probability factor. This can be interpreted as an average of predictions using an ensemble of networks. Dropout can also help in stabilizing the performance with regard to computation by removing additional synaptic connections (synaptic weights) that are not helping much towards the model's decision.

Exercise 4.3.1.j. Use dropout in the hidden layers for regularization.
Exercise 4.3.1.k. Use grid search or random search to figure out the optimal value of hyperparameters.

MNIST looks like a trivial problem for Neural Networks to solve. Let us consider a more complex problem of real world objects – the CIFAR-10 dataset. The dataset contains colored images from 10 classes belonging to real world objects.

Exercise 4.3.1.l. Train the basic MLP classifier in TensorFlow on the CIFAR-10 dataset and obtain the accuracy.

4.3.2. Convolutional Neural Networks

As we saw in the previous section, MLP overfits the image data if there is no feature engineering and the model is just given raw pixels. Convolutional Neural Net[42] (CNN) deal with this problem in an elegant way.

We shall consider the running problem of classification on the CIFAR-10 dataset. A CNN has two parts – convolutional filters and pooling operations that act as feature extractors from the raw input and a standard MLP which acts as the classifier. Both components are trained end-to-end using a single loss function via error back propagation.

Each convolutional layer has the following hyperparameters – number of filters, size of filter, stride and padding. Padding with zeros is done to make the dimensions compatible and prevent the output dimensions from shrinking. Pooling layer downsamples the data representation by averaging or taking the max of values over a region.

Exercise 4.3.2.a. Train a CNN (similar to AlexNet or VGGnet) in TensorFlow for the CIFAR-10 dataset using convolution with padding and pooling operations.
Exercise 4.3.2.b. Add dropout to the network in the above exercise.
Exercise 4.3.2.c. Plot the loss function with respect to the iteration number using TensorBoard.
Exercise 4.3.2.d. How does the filter size, number of filters and the depth of the network affect accuracy?
Exercise 4.3.2.e. How well does the network perform without the pooling layers where down sampling is done by using a bigger stride and no padding?

A lot of success of training depends on how well the weights are initialized. Initializing the weights as zero doesn't work as it doesn't break the symmetry. Initializing with small random numbers leads to smaller gradients. Xavier initialization and He et al. [link/citation required] initialization are two most popular initialization techniques – available in most deep learning libraries.

Exercise 4.3.2.f. Instead or initializing the weights with small random numbers, use Xavier or He et al. weight initialization techniques.

A technique to improve the flow of gradients through the network and in principle, allowing training deeper networks, is batch normalization. It explicitly forces the activations throughout a network to take on a unit Gaussian distribution and is added as a layer after FC/Conv layer but before the nonlinearity.

Exercise 4.3.2.g. Add batch normalization to the network layers and report the accuracy.

Apart from batch normalization for a better gradient flow, Google and Microsoft introduced various network architectures that improve the flow of gradients in the network and hence enhancing learning in the lower layers.[should we provide some links or citations to these articles from Google and MS?]

Exercise 4.3.2.h. Use Inception module (similar to GoogleNet) for the convolutions and report the results.
Exercise 4.3.2.i. Use skip connections in the network (similar to ResNet) and report the accuracy.

[42] Stanford CS231n Convolutional Neural Networks for Visual Recognition (Winter 2016)

When the amount of training data is less, it is usually a good idea to augment the data. Image data can be augmented by horizontal flips, random cropping and scaling and color jittering.

Exercise 4.3.2.j. Augment the training data using standard techniques and report the accuracy gain.
Exercise 4.3.2.k. Perform an ablation study to find out the impact for each module in the network.

4.3.3. Recurrent Neural Networks / LSTM / GRU

Consider the problem of the 20 News Group dataset. In Support Vector Machine (SVM), we had to do feature engineering to obtain the document representation. Recurrent Neural Networks (RNN) are deep learning models that allow us to learn the classification model just by word representations without doing any feature engineering.

A usual way to represent words is one-hot encoding. One hot representations are sparse and in principle don't represent anything meaningful or any relationship between words. According to the hypothesis that words occurring together in a context have similar meanings, we try to build a model called word2vec that converts the one-hot encoding to a dense vector where the vectors end up having a semantic relationship.

Two popular approaches to train word2vec are the Continuous Bag of Words (CBOW) model and the Skip Gram Model. In the CBOW model, the target word is predicted using the context words as input. The skip gram model is the opposite of CBOW, the context words are predicted using the target word as input. The prediction model is a simple softmax function with the cross-entropy loss. However, computing the softmax over the entire vocabulary for every context-target pair is expensive and we scale it up using noise contrastive estimation.

Exercise 4.3.3.a. Implement the word2vec algorithm in TensorFlow using CBOW model.
Exercise 4.3.3.b. Implement the word2vec algorithm in TensorFlow using skip-gram model.
Exercise 4.3.3.c. Visualize the embedding using t-SNE.

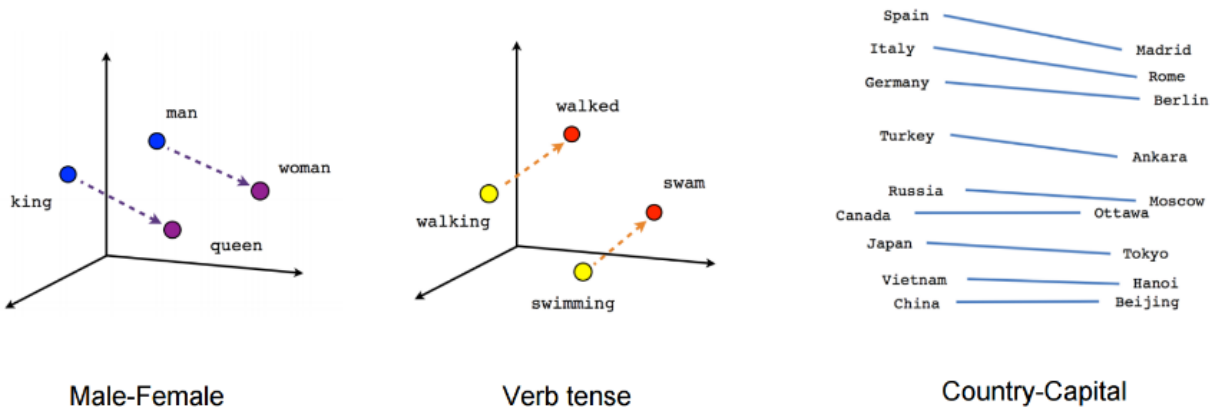


Figure 18 Word2Vec learnt semantic relationships between pairs of values

In the above diagram[43], we see that word2vec learnt semantic relationships like queen-king = woman – man. It also learnt how to relate countries to their capitals and different tenses of the words. These kind of relationships are absent in a one-hot encoding and are learnt in a totally unsupervised fashion. Unless

[43] TensorFlow Word2vec tutorial - https://www.tensorflow.org/tutorials/word2vec

we have a huge corpus, it is usually not a good idea to learn these embeddings. Pre-trained word2vec or GloVe embeddings for Wikipedia can be found online.

Treating the SVM results as a baseline – we shall now try to obtain the classification accuracy using a recurrent neural net. A RNN is capable of keeping information across time steps by maintaining an internal state. It can be thought of as multiple copies of the same network, each passing a message to a successor.
Exercise 4.3.3.d. Train a RNN in TensorFlow to classify the articles of the 20 News Group dataset.

RNNs fail to model long term dependencies due to the vanishing gradient problem because of which they mostly fail to give expected results. For example – a RNN can successfully model "Coffee is effective as it contains …" by completing the sentence with "caffeine". However, it would fail model something like "I just had coffee. I have to complete this assignment by tomorrow. I cannot … sleep". The dependence between sleep and coffee is not modeled by a RNN as the gradients become close to zero while back propagating through a larger number of time steps.

This problem can be dealt by using gated models like Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU), which avoid the repeated multiplication by a similar factor leading to a better flow of gradients and preventing them from vanishing quickly.

Exercise 4.3.3.e. Compare the performance of the RNN to that of the GRU and LSTM.
Exercise 4.3.3.f. Plot the loss vs iteration graph for the three cases and visualize it using TensorBoard.
Exercise 4.3.3.g. Add more layers to the LSTM / GRU model and obtain the accuracy.
Exercise 4.3.3.h. Train a single layered bi-directional LSTM / GRU to obtain the accuracy.
Exercise 4.3.3.i. Train a multi-layered bi-directional LSTM/ GRU and compare its performance.
Exercise 4.3.3.j. Add dropout to all the above models.

4.3.4. Autoencoder

Consider the Pascal VOC dataset – it has images from 20 classes with their bounding boxes. We have already seen one way to obtain feature representations – by training a CNN for classification and taking the feature map and the end of the convolutional layers. This feature map can now be used for other downstream tasks – such as localization of the class (obtaining the bounding box). The features obtained by this method are class specific features.

Exercise 4.3.4.a. Train a CNN classifier for the images in Pascal VOC and obtain the feature map. Now use these features to train a MLP with a regression head (no softmax at the output layer) and obtain the localization error.

Suppose the class labels for the images were not available and we wanted to learn a feature representation for the images in a totally unsupervised fashion. We can do this by trying to model the identity function by first obtaining an intermediate feature representation using convolutional layers and then trying to regenerate the original image from this representation by using upconvolution

(reverse of convolution). The entire network can be trained end to end using back propagation. This is called an *autoencoder*.

Exercise 4.3.4.b. Train an autoencoder in TensorFlow using the unlabeled examples to learn the identity function with the encoder and decoder having convolutional and upconvolutional layers respectively.
Exercise 4.3.4.c. Instead of having different weights for the encoder and decoder, use weight sharing in the convolutional and upconvolutional layers.
Exercise 4.3.4.d. Obtain two representations, one using L1 and another using L2 loss for training.

Now that we have an unsupervised class agnostic representation, we can see how good the representation is by using it for our downstream task of localization.

Exercise 4.3.4.e. Evaluate each of the representations by removing the decoder and adding in fully connected layers with a regression head to obtain the localization error.
Exercise 4.3.4.f. Compare the localization error for the class agnostic and the class specific representations.

4.3.5. Siamese

Consider the Quora Question Pairs dataset – given two questions on Quora, the task is to flag them as duplicates or not. The dataset is available on Kaggle - https://www.kaggle.com/c/quora-question-pairs

As we have seen, LSTM should be a good model for this kind of task. Here, we shall use two LSTMs with shared weights and then try to make the prediction for duplicates.

Exercise 4.3.5.a. Obtain the word2vec embeddings for each question in the dataset.

A Siamese network is a combination of two networks which have the exact same weights (share weights) and each network produces an output representation for its input. The output representations are then matched with the similarity of the inputs and the entire network is trained end-to-end with backpropagation performing simultaneous weight updates as the weights are shared.

Exercise 4.3.5.b. Set up a Siamese network in TensorFlow consisting of two LSTMs with weight sharing taking inputs as the two sentences.
Exercise 4.3.5.c. Train the two LSTMs by computing the similarity between the outputs for the two sentences and using the duplicate label field as the target.
Exercise 4.3.5.d. Plot the loss vs iteration graph using TensorBoard and report the accuracy.

5. Projects

5.1.1. Sign Language and Gesture Recognition

Introduction:

Sign Language and Gesture Recognition are open problems in the area of Machine Vision, a field of Computer Science that enables systems to emulate human vision. Gesture recognition has many applications in improving Human-Computer Interaction (HCI), by enabling users to interact in a more natural manner with computers. It is also particularly useful in the field of Sign Language Translation, wherein a video sequence of symbolic hand gestures is translated into natural language. The use of these algorithms on an embedded device could help the hearing impaired communicate more effectively with others. The goal of the ML project is to build such a system to classify static (i.e. no motion involved) gestures and evaluate its performance.

Problem Statement:

Build a model to solve the following problem:

Given an image of a person's upper body and hand, locate the hand and classify the gesture.

Dataset:

You will be given a dataset of images which represent the different letters of the alphabet. These images contain the hand and part of the upper body of a person making different gestures. There will be several thousand images for you to train on. They have been collected from 20 different native signers and part of this dataset will be used for training your classifiers and localizers. This dataset contains a CSV file with the locations of the bounding box of the hand in each image. You can use this data to crop the images and isolate the hand region, which in turn will be used to build the classifier. This data of bounding boxes constitutes the ground truth, and will be used to evaluate the performance of your localization algorithm as well.

5.1.2. Google Street View House Numbers (SVHN)

Introduction:

You have been given images of house numbers from Google Street View data. Your task is to use computer vision and machine learning to extract the house number sequence correctly from an image.

The problem:

As in the real world, the length of the sequence of digits in a house number is variable. To add to the complexity of the problem, the images are not of the same size and the house numbers don't occupy the entire image. You need to tell the digits occurring in each image while maintaining the sequence - so that you can tell the house number correctly.

Dataset:

Training:

The training data consists of 33402 images. The images are accompanied by a CSV file which contains the label and bounding box specification for each digit in the image.

Test:

The test data will consist of images that are already cropped. Hence, a test image will be of a similar size as the ground truth bounding box of the entire sequence of digits.

5.1.3. General Back propagation Implementation

In this assignment, you are expected to create a simple artificial neural network library using Python and Numpy. This library should support ANNs of arbitrary size as defined by the user. It should support Regression and Classification tasks with the following specifications:

Activation functions:

1. Sigmoid
2. ReLU
3. Softmax
4. Linear (No activation, only linear transform)

Loss functions:

1. L1 Loss
2. L2 Loss
3. Cross Entropy
4. SVM Loss

Optimizers:

1. SGD
2. Momentum

As you might have realized, you are **required to use a computational graph** approach for this problem for the sake of modularity and flexibility.

Implementation Specifics:

There will be a user facing DenseNet class:

```
class DenseNet:
   def __init__(self, input_dim, optim_config, loss_fn)
      #Initialize the computational graph object.

   def addlayer(self, activation, units)
      #Modify the computational graph object by adding a layer of the specified type.

   def train(self, X, Y):
      """
      This train is for one iteration. It accepts a batch of input vectors.
      It is expected of the user to call this function for multiple iterations.
      """
      return loss_value

   def predict(self, X):
      """
      Return the predicted value for all the vectors in X.
      """
      return predicted_value
```

Recommended API for the implementation of the computational graph and the computational gates.

```
class Graph:
   # Computational graph class
   def __init__(self, input_dim, optim_config, loss_fn):
      pass

   def addgate(self, activation, units=0):
      pass

   def forward(self, input):
      return predicted_value

   def backward(self, expected):
      return loss_val

   def update(self):
      pass


class ReLU:
   # Example class for the ReLU layer. Replicate for other activation types and/or loss functions.
   def __init__(self, d, m):
      pass

   def forward(self, input):
      return gate_output

   def backward(self, dz):
      return gradients_wrt_inputs
```

```
class Optimizer:
  def __init__(self, learning_rate, momentum_eta = 0.0):
    pass
```

5.1.4. Sematic Segmentation for RBCs

Introduction:

Segmentation tasks in computer vision have been attracting a lot of attention recently. We will deal with a specific type of segmentation called semantic segmentation. Semantic segmentation means assigning a class to each pixel in an image.

The problem:

You have been given variable sized images of blood cells and your task is to use computer vision and machine learning to segment the RBCs. This means assigning 1 to each pixel that makes up a RBC in the image and 0 otherwise (called background).

Dataset:

Training:

The training data consists of 169 variable sized images. Each image is accompanied by a mask which is the ground truth for that segmentation problem. Please note that each image could have more than 1 RBC.

Test:

The test data consists of variable sized RGB images with each image possibly having more than 1 RBC.

Tirtharaj's review and comments:

I think, the deep learning portion needs to be converted in a slightly more detailed fashion rather than just explaining in paragraphs of sentences. Further, we need to add a lot of references, and it will help students getting into more research (mathematics) oriented details.