Python Machine-Learning Book

Python libraries:

NumPy, Matplotlib, scikit-learn, Pandas

**1 Giving Computers the ability to learn from data.**

**Machine Learning:**

* Supervised Learning
  + Learn a model from **labeled training data** . The desired output signals (labels) are already known.
    - Discrete. Class labels. Outcome signal is a classification task.
    - Continuous. Regression. Outcome signal is a continuous value.
  + Classification for prediction class labels. [Discrete]. Assign categorical, unordered labels to instances. Ex:
    - Binary classification: spam-nonspam model
      * A 2-dimensional dataset means that each sample has two values associated with it: x1 and x2.
    - Multi-class classification: Handwritten character recognition.
  + Regression for predicting continuous outcomes. [Continuous]. We are given a number of predictor (explanatory variables) X and a continuous response variable (outcome) Y and we try to find a relationship between those variables.Ex:
    - Given X and Y we fit a straight line to this data that minimizes the distance (optimal problem) - most commonly the average squared distance.
* Reinforcement learning
  + The goal is to develop a system (agent) that improves its performance based on interactions with the environment. *The state of the Agent affects the Environment and this returns to the agent with an Action and a Reward.* The agent will try to learn a series of actions that maximizes this reward via an exploratory trial-and-error approach or deliberative planning. Ex:
    - Chess game. The agent decides upon a series of moves depending on the state of the board (environment) and the reward can be defined as win or lose at the end of the game. *Here the reward signal is delibered a while after each action or move.*
* Unsupervised Learning. Discovering hidden structures.
  + We are dealing with unlabeled data or data of unknown structure. We will explore the structure of our data to extract meaningful information without the guidance of a known structure (supervised learning) or reward function (reinforced learning).
  + Finding subgroups with clustering (aka unsupervised classification). To organize data into meaningful subgroups withot any prior knowledge of their group memberships.
  + Dimensionality Reduction for data compression. Often dealing with high dimensionality data (each observation comes with a high number of measurements). *DR is used for preprocessing: to remove noise from data, compressing the data onto a smaller dimensional subspace while retaining most of the relevant information*. *Also useful for visualizing data (high dimensional data can be reduced into 3D or 2D scatterplots or histograms.*

**Terminology**:

ML uses vector-matrix notation.

Sample (rows in the matrix-superscript); Features (columns in the matrix-subscript); Measurement (value on each position).

**A Roadmap for building machine learning**

*(Important parts of a machine learning system accompanying the learning algorithm.)*

Typical workflow diagram for predictive modeling:

Machine generated alternative text:
Feature Extraction and 
Feature Select-ion 
Dimensionality Reduction 
Sampling 
Training Dataset 
Test Dataset 
Preprocessing 
Learning 
AlgMithrn 
Learning 
Final Model 
Evaluation 
Prediction 
Model Selection 
Cross-V i 
Performance Metrics 
Hyperparameter Optimization 

* Preprocessing. Crucial stage.
  + Many machine learning algorithms require that the selected features are on the same scale for optimal performance, often achieved by transforming the features in the range [0,1] or a standard normal distribution with zero mean and unit variance.
  + Some selected features may be highly correlated and therefore redundant -> Dimensionality reduction (less storage space) much faster learning algorithm.
  + To determine the performance of our algorithm the dataset can be randomly divided into training (to train and optimize out machine learning model) and test set (to evaluate the final model).

**Training and selecting a predictive model**

Many machine learning algorithms have been developed to solve different problem tasks.

Hyperparameter optimization techniques.

There are not universal models that can be applied to every scenario. One should be very careful when picking a model. Also, the model default parameters may be not optimal for our data, therefore we need to fine tune them (knobs analogy).

**Evaluating models and predicting unseen data instances**

The parameters for the previously mentioned procedures—such as feature scaling and dimensionality reduction—are solely obtained from the training dataset, and the same parameters are later re-applied to transform the test dataset, as well as any new data samples—the performance measured on the test data may be overoptimistic otherwise.

**Python for machine learning**

Although the performance of interpreted languages, such as python, for computation-intensive tasks is inferior to lower-level programming languages, extension libraries such as ***NumPy*** and ***SciPy*** have been developed that build upon lower layer Fortran and C implementations for fast and vectorized operations on multidimensional arrays. ***Scikit-learn*** Open source machine learning library.

**2\* Training Machine Learning Algorithms for Classification**

One of the first algorithmically described machine learning algorithms for classification:

* Perceptron
* Adaptive linear neurons

Points covered in the chapter:

* Building an intuition for machine learning algorithms
* Using pandas,Numpy and matplotlib to read in , process and visualize data
* Implementing linear classification algorithms in Python

**Artificial neurons- early story of machine learning**

Neurons are interconnected nerve cells : processing and transmitting signals.

Perceptron learning: Rosenblatt proposed an algorithm that would automatically learn the optimal weight coefficients that are then multiplied with the input features in order to make the decision of whether a neuron fires or not.

Supervised learning+Perceptron algorithm= predict if a sample belongs to one class or the other.

* **Perceptron: creates a threshold to make the differentiation. (it is a single layer neural network)**

This *thresholded* perceptron model uses a reductionist approach to mimic how a single neuron in the brain works: it either fires or it doesn’t.

z=w1x1+w2x2+…wmxm

Sigma(z)={ 1 if z>= theta (threshold); -1 otherwise

*Perceptron steps:*

1. Initialize the weights to 0 or small random numbers
2. For each training sample perform the following steps
   1. Compute the output value y
   2. Update the weights

Updating weights:

w\_j:=w\_j+Delta w\_j

Perceptron learning rule: Delta w\_j= \eta (y^I - \hat{y}^i)\*x\_j^I

\eta is the learning rate 0.0-1.0

\*All the weights are updated simultaneously

Machine generated alternative text:
AW2 
output( ) 
output 

The "push" is directly proportional to the value of the measure.

THE CONVERGENCE OF THE PERCEPTRON is only guaranteed if the two classes are linearly separable and the learning rate is sufficiently small. In the opposite case we can set a maximum number of passes over the training dataset (epochs) and/or a threshold for the number of tolerated misclassifications- otherwise the perceptron would never stop updating the weights.

Machine generated alternative text:
Now, before we jump into the implementation in the next section, let us summarize 
what we just learned in a simple figure that illustrates the general concept of 
the perceptron: 
Weight update 
Error 
Net input 
function 
Output 
1 
Activatio n 
function 

The perceptron algorithm is not only used for binary cases but also on multiclass problems : One-vs-All / One-vs-Rest.

PERCEPTRON LIMITATIONS: Convergence. Frank Rosenblatt proved mathematically that the perceptron learning rule converges if the two classes can be separated by a linear hyperplane. Otherwise the weights will never stop updating unless we set a maximum number of Epochs.

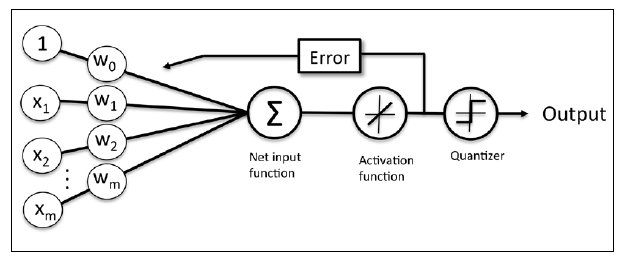
“Intuitively we can think of the reason as the weights are continuously being updated since there is always at least one misclassified sample present in each epoch.

* **ADAptive LInear NEurons and the convergence of learning (Also a single layer neural network)**

Adaline. This algorithm illustrates the key concept of defining and minimizing cost functions (foundations of more advanced ML algo: logistic regression and other regression models).

In Adaline, weights are updated based on a linear activation function rather than a unit step function (as in perceptron). Activation function is simply the identity function of the net input so that:

|  |  |
| --- | --- |
| **Activation Function** | |
| Adaline: Identity function | Perceptron: Step function |
|  | |
|  |  |



Adaline: Flowchart

* + Activation function is used for learning the weights
  + Quantizer is used to predict the class labels

***Big difference is that to compute the model error and update the weights we are now using the continuous linear output from the Activation function rather than the binary class labels (remember than in perceptron y was just 1 or -1).***

**ADALINE: Minimizing cost functions with gradient descent (taking a step into the opposite direction of the gradient)**

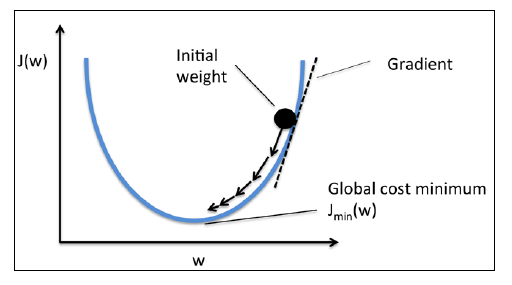
**\*\*Question: does this guarantee the global minimum??? Imagine a plot with more that one valley, once one is achieved, the gradient will be 0 therefore no need to take any steps??\*\***

One of the key ingredients of supervised machine learning algorithms is to define an objective function that is to be optimized during the learning process (analog to minimizing a cost function).

|  |
| --- |
| **Adaline Cost function:** Sum of Squared Errors (SSE) between the calculated outcome and the true class label. |
| Gradient descend: the gradient is calculated from **the whole training data (batch gradient descend) – Computationally very costly, not suitable for large scale datasets** |

Advantages of this cost function:

* Continuous (Differentiable) and Convex. Mathematically it is guaranteed the existence of weights that minimize this cost function. This can be done using: then *Gradient descent* algorithm.



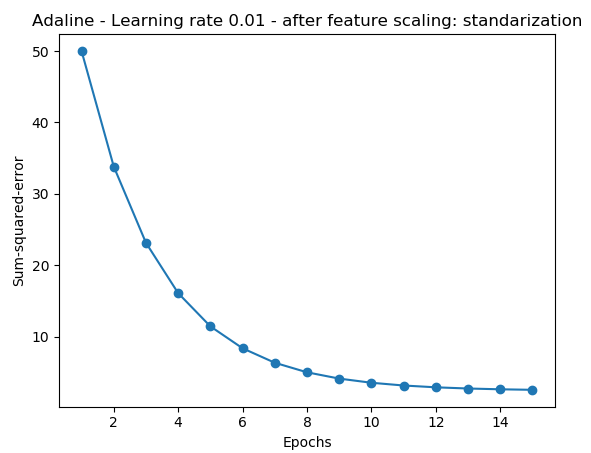
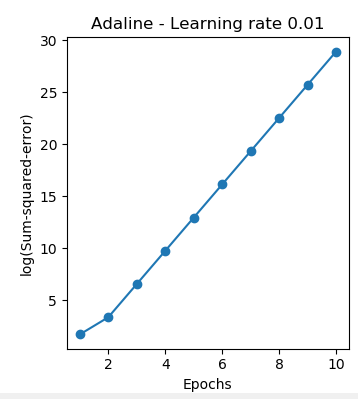
* + - * + Variables: Value of the learning rate (like eta on perceptron) and the slope of the gradient (how fast, the values move)

Although the learning rule looks identical as the Perceptron rule, the activation function is a real number now and not an integer class label. Furthermore, the weight update is calculated based on all samples in the training set (instead of updating the weights incrementally after each sample), reason why this approach is referred as BATCH gradient.

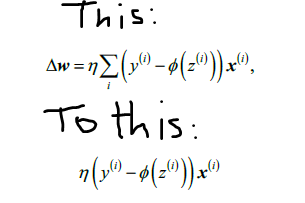
LEARNING RATE-> Eta and NUMBER OF EPOCHS-> n\_iter are the so-called **HYPERPARAMETERS**. There are some techniques to automatically find the values of this that yield to optimal performance for the classification model.

Many algorithms require some sort of ***feature scaling*** for optimal performance-> The so-called Machine Learning Classifiers (Scikit-learn). Here we will use one called **standardization:** transform the data into a standard normal distribution.

*Observation from ex1: The following plot is the result of using feature scaling on the Adaline algorithm under the use of the same Hyperparameters. On the left the error diverges, overshoots, one of the possible causes is a very large learning rate (it is proved that lowering down the learning rate makes the solution converge, also in the simulation). On the right, same learning rate + feature scaling makes the solution converge (although this SSE doesn’t go to zero). Feature scaling helps the algorithm to achieve optimal performance. Standarization just removes the trend of the data (moves the mean to zero) and transforms the standard deviation to 1.*



**Large Scale machine learning and Stochastic Gradient Descent (alternative to gradient descent)**



* Weights are updated incrementally for each training sample (not the whole data set).
* Faster convergence (more frequent weight updates).
* Noisier error surface, makes it easier for the algorithm to escape shallow local minima.
* Necessary to present data in a random order, shuffle of the training set in every epoch is required to prevent cycles.
* Can be used online, on-the-fly as new data arrives. The whole dataset is not required.

**Mid way solution: GD-mini-batch learning. The batch size is limited to a fixed amount. Advantages are:**

* + - * Faster convergence than GD. Weights are updated more often
      * Computationally less intensive than SGD. For loops (each iteration is a loop) can be replaced to vectorial operations (more efficient).

SUMMARY:

* Basic concepts of linear classifiers for supervised learning.
* Perceptron
* Adaline
* Next episode: Scikit-learn

**3\* A TOUR OF MACHINE LEARNING CLASSIFIERS USING SCIKIT-LEARN**

* + - * Introduction to the concepts of popular classification algorithms
      * Using the scikit-learn machine learning library
      * Questions to ask when selecting a machine learning algorithm

**Choosing a classification algorithm**

Different machine learning algorithms depending on the number of features or samples, the amount of noise in a dataset, and whether the classes are linearly separable or not.

Training a machine learning algorithm:

1. Selection of features
2. Choosing a performance metric
3. Choosing a classifier and optimization algorithm
4. Evaluating the performance of the model
5. Tuning the algorithm

**First steps with scikit-learn**

Contains a UI. Machine learning algorithms and preprocessing tools (~~somehow like the SystemIdentification tool in matlab).

**Training a perceptron via scikit-learn**

Good practice: class labels stored as integers

To evaluate how well a trained model performs on unseen data, we split the dataset into: training and test sets. Although this will be covered in chapter 5: Compressing data via dimensionality reduction.

Most algorithms in scikit-learn support multiclass classification by default via the One-vs-Rest (OvR) method.

As we knew perceptron is not a good machine learning algorithm if the dataset is not perfectly linearly separable.

To learn more about the scikit-learn library: <http://scikit-learn.org/stable/> . Here we can find that these kind of neuron models have other parameters than n\_iter, eta0, random\_state. These are omitted in the examples for clarity.

**Modeling class probabilities via logistic regression**

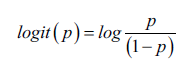
It is a simple and powerful algorithm for linear and binary classification problems: logistic regression. (This is a classification method)

**Logistic regression intuition and conditional probabilities**

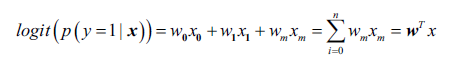
First time that we are using probabilities…

Performs very well on linearly separable classes.

Logit function (the logarithmof the odds ratio):



This function takes the input values in the range 0 to 1 and transforms them to values over the entire real number range, which we can use to express a linear relationship between feature values and the log-odds.

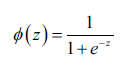
=z (net input) linear combination of weights and sample features

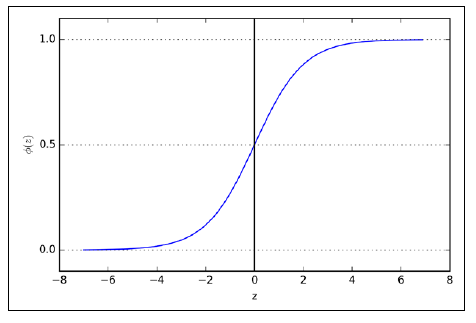
Now what we are actually interested in is predicting the probability that a certain

sample belongs to a particular class, which is the inverse form of the logit function. It

is also called the logistic function, sometimes simply abbreviated as sigmoid function

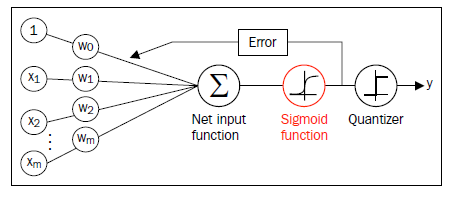
due to its characteristic S-shape.



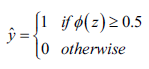
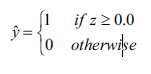


Sigmoid approaches to 1 if z goes towards infinity (z to inf). Similarly goes towards 0 for z goes towards minus inf. Therefore this function (continuous) will take real values on the interval [0, 1] with an interception at Sig(z)=0.5. The output of the sigmoid function then is the probability of particular sample belonging to class 1, given its features x parameterized by the weights w.

Logistic regression figure:



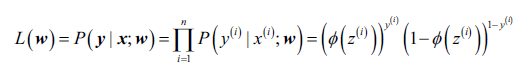
The predicted probability can then simply be converted into a binary outcome via a quantizer (unit step function):

 equivalent to 

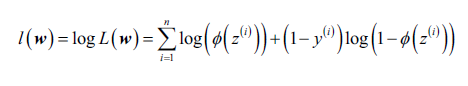
Logistic regression is used in weather forecasting, not only to report if it is going to rain but also to report the chance of rain. Logistic regression can be also used to predict the chance that a patient has a particular disease given certain symptoms…

**Learning the weights of the logistic cost function**

First let’s define the likelihood L that we want to maximize when we build a logistic regression model, assuming that the individual samples in our dataset are independent of one another. The formula is as follows:

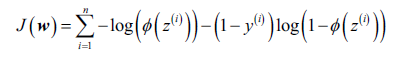


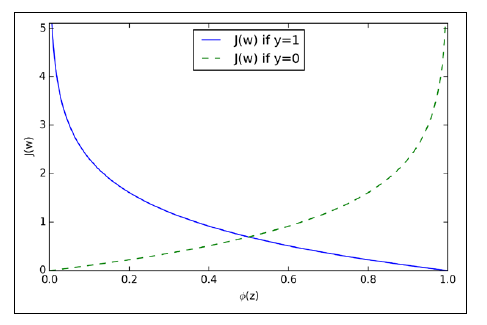
Easier to maximize the log-likelihood function:



* 1. Log reduces the potential numerical underflow that may occur when likelihoods are very small.
  2. Multiplications can be transformed into sumations which makes it easier to obtain the derivative

Rewriting the log-likelihood function into a cost function that can be minimized using gradient descent.





**Training a logistic regression model with scikit-learn**

**Lessons Learned**:

Class labels: y

Classification methods (and their activation functions):

* Perceptron. Step activation
* Adaline. Linear activation
* Logistic regression

Reinforcement Learning limitations:

**Tackling overfitting via regularization**

Overfitting:

* a model performs well on training data but does not generalize well to unseen data (test data).
* A model has a high variance, has too many parameters

Underfitting (high bias):

* The model is not complex enough to capture the pattern in the training data well and therefore also suffers from low performance on unseen data.

Variance measures the consistency (or variability) of the model prediction for a particular sample instance if we would retrain the model multiple times, for example, on different subsets of the training dataset.