Outline:

1. Simple Perceptron
2. Multilayer Perceptron
   1. Very sensitive to feature scaling, should always scale data
   2. Finding a reasonable regularization parameter (alpha) is best done using [GridSearchCV](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#sklearn.model_selection.GridSearchCV), usually in the range 10.0 \*\* -np.arange(1, 7).
3. POS identification in text:
   1. “The Perceptron tagger has resulted the highest accuracy, 88.7%. This tagger is the default NLTK’s POS tagger since NLTK version 3.1. The Perceptron tagger, also known as Averaged Perceptron Tagger is ported from TextBlob Perceptron Tagger into NLTK and is implemented originally by Matthew Honnibal. It is a pre-trained model on Penn Treebank Wall Street Journal (WSJ). This tagger is based on Hidden Markov Model (HMM) where next state is dependent only on the current state not on the previously occurred observation and which makes viterbi decoding possible. In simple words, it predicts the tag of the word on the basic of currently tag word on the previously tagged words”

Useful Links:

<https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html>

<https://scikit-learn.org/stable/modules/neural_networks_supervised.html> \*

<https://scikit-learn.org/stable/auto_examples/linear_model/plot_sgd_comparison.html#sphx-glr-auto-examples-linear-model-plot-sgd-comparison-py>

<https://www.nltk.org/_modules/nltk/tag/perceptron.html>

Averaged Perceptron Tagger: <https://iopscience.iop.org/article/10.1088/1742-6596/2224/1/012001/pdf>

To cite this article: Thing Thing Goh et al 2022 J. Phys.: Conf. Ser. 2224 012001

<https://medium.com/@ilyurek/perceptron-model-the-foundation-of-neural-networks-4db25b0148d>

|  | ***Simple Perceptron*** | ***Multilayer Perceptron*** |
| --- | --- | --- |
| ***How to implement*** | Linear Model, Perceptron | Non linear, neural network, |
| ***Regression*** | Not able to do regression | Lots of perceptrons can predict continuous variables well |
| ***Classification*** | Binary Classification | Can make better classifications as you increase the number of perceptrons and layers |
| ***Regularization*** | L1 and L2 regularization allowed for perceptrons   * *Penalty:* The penalty (aka regularization term) to be used. * *alpha:* Constant that multiplies the regularization term if regularization is used. | Regularization in multilayer perceptrons is key to inhibiting overfit. As the regularization alpha increases the variance can decrease |
| ***Stopping Criterion*** | * *tol:* The stopping criterion. If it is not None, the iterations will stop when (loss > previous\_loss - tol). * *max\_iter:* The maximum number of passes over the training data (aka epochs). It only impacts the behavior in the fit method * *n\_iter\_no\_change:* Number of iterations with no improvement to wait before early stopping. |  |
| ***Training*** | Stochastic gradient descent | Backpropagation  (Essentially gradient descent to maximize some error metric.) Same process as logistic regression, but utilizes the chain rule for derivatives to find gradient of the weights over all the layers.  Classification: cross entropy error  Regression: squared error |

Script: Jayce H

MultiLayered Perceptron or a Neural Networks

Multi Layered perceptrons help us to solve nonlinear, multiclass classification and regression problems.

A neural network can be diagrammed using a basic graph structure. With nodes representing input, activation functions and our output and edges representing the weight that is applied to each input from the previous layer.

Here are some specific parameters to keep in mind while using MLP’s:

* **Hidden layer sizes**, which designates the number of hidden layers and the number of nodes or neurons in each layer
* **Activation**: The function used for each of the nodes to
  + **Relu** piecewise function
  + **Logistic** sigmoid function
  + **Identity**: line with slope 1
  + **Hyperbolic Tangent (tanh(x))**
* **Solver**: The process used to compute the optimal weights to produce output y
  + **Lbfgs** - is an optimizer in the family of quasi-Newton methods. Used for smaller datasets to find weights to minimize error.
  + **Sgd(Stochastic Gradient Descent)** - Sort, train on one sample, update weights based on error, repeat for all samples and until you feel satisfied, an algorithm that updates weights iteratively improving computation speed with estimation
  + **Adam** - is an optimizer based on sgd that works for large datasets. Differing from sgd, it dynamically finds individual learning rates instead of a single learning rate per iteration like in sgd.
* **Batch**: changes the batch size of sgd and allows you to use only a subset of data to improve performance
* **Learning rate**: how quickly sgd can find the minimum of value of the functions. Smaller values make it take longer, you won't miss the minimum. Larger values are quicker but you could surpass the minimum and never converge to it properly

As with other ml models, regularization can help to reduce overfit on the data and make the model more generalized. This can be seen in the plots shown, with increasing the alpha parameter.

Now let’s take some time to explore how MLP can be used for Regression and Classification problems!