Capstone:
Classifying Mosquitos Carrying the West Nile Virus in Chicago

Author: Christopher Broll
Course: DATS 6501_01
Professor: Dr. Amir Jafari
Date: December 3, 2018

Introduction

In an effort to eradicate the West Nile virus from Chicago, in 2002 the Chicago Department of Public Health (CDPH) installed a monitoring/surveillance system to trap mosquitos and test them for the virus. In 2015 they posted a competition on *Kaggle.com*, calling for a model that classifies mosquitos that carry it. With datasets that include the distribution of pesticides, weather conditions, and the training and test datasets, this model would be used to distribute/spray pesticides in areas at high risk. Since many solutions did not take a neural networking approach, I aim to develop a comparative study between the models on the top of the leaderboard and a multilayer perceptron (mlp) that I designed for binary classification as methods to classify mosquitos that carry the West Nile virus in Chicago. This paper will *describe the datasets we used, the deep learning network and training algorithm, experimental setup, results and summary/conclusion.*

Description of the Dataset

There are three datasets provided on *Kaggle*, *train.csv*, *weather.csv* and *spray.csv*. The train dataset, which is the primary dataset, includes features related to mosquito traps in Chicago from 2007, 2009, 2011 and 2013. The following is a list of its features and their respective descriptions:

- *Id*: the id of the record
- *Date*: date that the WNV test is performed
- *Address*: approximate address of the location of trap. This is used to send to the GeoCoder.
- *Species*: the species of mosquitos
- *Block*: block number of address
- *Street*: street name
- *Trap*: Id of the trap
- *AddressNumberAndStreet*: approximate address returned from GeoCoder
- *Latitude, Longitude*: Latitude and Longitude returned from GeoCoder
- *AddressAccuracy*: accuracy returned from GeoCoder
- *NumMosquitos*: number of mosquitoes caught in this trap
- *WnvPresent*: whether West Nile Virus was present in these mosquitos. 1 means WNV is present, and 0 means not present.

The weather dataset contained weather reports from two stations, station 1 and station 2, which are *Chicago O'Hare International Airport* and Chicago *Midway International Airport*, respectively, from the 2007 to 2014. Below is a list of its features and their respective descriptions:

- *Station*: station number of weather report
- *Date*: date of weather report
- *Tmax*: maximum temperature Fahrenheit (F)
- *Tmin*: minimum temperature F
- *Tavg*: average temperature F
- *Depart*: departure from normal F

- *DewPoint*: average dew point F
- *WetBulb*: average wet bulb F
- *Heat*: 65 -Tavg, when Tavg below base 65 Degrees F
- *Cool*: Tavg - 65, when Tavg above base 65 Degrees F
- *Sunrise*: calculated time of sunrise from 24 hour clock
- *Sunset*: calculated time of sunset from 24 hour clock
- *CodeSum*: Coded weather phenomena
- *Depth*: snow/ice on ground inches
- *Water1*: water equivalent
- *SnowFall*: snowfall in inches
- *PrecipTotal*: rainfall in inches
- *StnPressure*: average station pressure in inches of hg (mercury/barometer)
- *SeaLevel*: average sea level pressure
- *ResultSpeed*: resultant windspeed in mph
- *ResultDir*: resultant wind direction (whole degrees)
- *AvgSpeed*: averaged windspeed in mph

The spray dataset contains locations and dates from 2011 and 2013 where/when pesticides were distributed in Chicago. Its features and their respective descriptions are listed below:

- *Date, Time*: the date and time of the spray
- *Latitude, Longitude*: the Latitude and Longitude of the spray

The Deep Learning Network and Training Algorithm

In the framework *PyTorch*, the network that I designed is a multilayer perceptron, a *feed-forward* network with *backpropagation*, which means that one feeds the features and targets into the input layer, calculates the error with respect to a performance index and optimizer, and then updates the weights and biases accordingly. This network has five layers, an input layer with a transfer function, three hidden layers and an output layer with a transfer function. The input size of the network is eight (eight features), and hidden layer one, two and three have seven, five and three nodes, respectively.

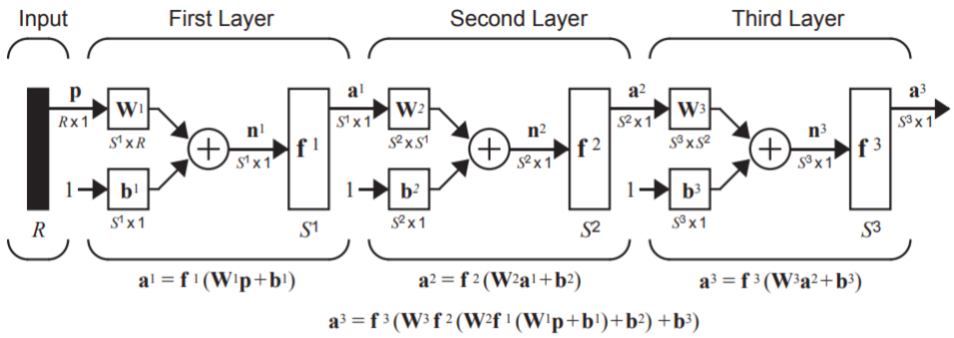


Figure 11.7 Three-Layer Network, Abbreviated Notation

Because of the utility of its derivative, the function *ReLU* is the transfer function for the input layer and hidden layers. In particular, its derivative is a function called *hardlim*, which is equal to 1 when the input is greater than zero and equal to zero when the input is less than zero. This attribute is attractive because other transfer functions suffer from vanishing gradients that stops the network from learning. For the output layer, we use the function hyperbolic tangent or *tanh* (*tansig* below).

Name	Input/Output Relation	Icon	MATLAB Function
Hard Limit	$a = 0 \quad n < 0$ $a = 1 \quad n \geq 0$		hardlim
Symmetrical Hard Limit	$a = -1 \quad n < 0$ $a = +1 \quad n \geq 0$		hardlims
Linear	$a = n$		purelin
Saturating Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n \leq 1$ $a = 1 \quad n > 1$		satlin
Symmetric Saturating Linear	$a = -1 \quad n < -1$ $a = n \quad -1 \leq n \leq 1$ $a = 1 \quad n > 1$		satlins
Log-Sigmoid	$a = \frac{1}{1+e^{-n}}$		logsig
Hyperbolic Tangent Sigmoid	$a = \frac{e^{n}-e^{-n}}{e^{n}+e^{-n}}$		tansig
Positive Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n$		poslin
Competitive	$a = 1$ neuron with max n $a = 0$ all other neurons		compet

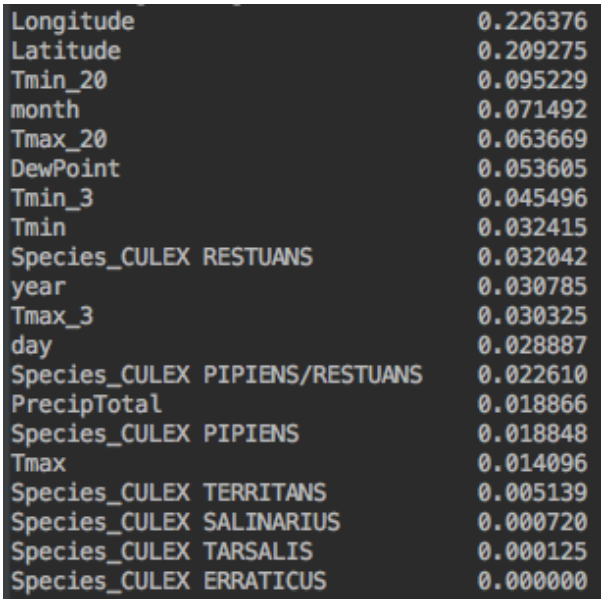
Table 2.1 Transfer Functions

We chose to use this transfer function because it is bounded with -1 for negative inputs (false for classification) and +1 for positive inputs (true for classification). Since *CrossEntropyLoss* is our performance index, which requires a node for each class, we have two nodes in our output layer, where [1, 0] is class one and [0, 1] is class zero. This decision is made after the training where the larger node determines the class.

Experimental Setup

Following *doyleax's* preprocessing (GitHub account) to build a comparative study, we preprocessed the datasets accordingly: First, we grouped the features *trap*, *longitude*, *latitude*, *date*, *species* and *wnv_present*, aggregating by the sum of *NumMosquitos* in *train.csv*. This enabled us to quickly create a dataframe that contained the presence of wnv for each species of mosquitos at a given time and location (trap). Secondly, we preprocessed weather.csv, additionally, averaging temperature, rainfall and dew point across a specified length of time for each observation. And finally, since *spray.csv* only covered years 2011 and 2013, this was excluded from the experiment. Continuing to follow *doyleax*, we joined the preprocessed *train.csv* and *weather.csv* by date and subsequently trained a random forest with this data for

feature importance. This feature importance, in addition to other feature importances included as attributes from other models, we selected the features *Longitude*, *DewPoint*, *Tmin_20*, *Species_CULEX PIPENS*, *Latitude*, *Tmax_20*, *Tmin_3* for training our model.

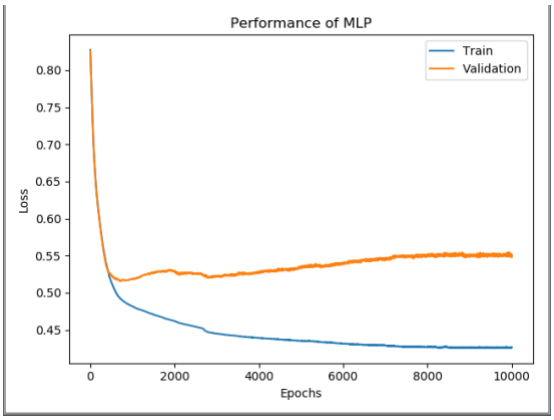


Longitude	0.226376
Latitude	0.209275
Tmin_20	0.095229
month	0.071492
Tmax_20	0.063669
DewPoint	0.053605
Tmin_3	0.045496
Tmin	0.032415
Species_CULEX RESTUANS	0.032042
year	0.030785
Tmax_3	0.030325
day	0.028887
Species_CULEX PIPIENS/RESTUANS	0.022610
PrecipTotal	0.018866
Species_CULEX PIPIENS	0.018848
Tmax	0.014096
Species_CULEX TERRITANS	0.005139
Species_CULEX SALINARIUS	0.000720
Species_CULEX TARSALIS	0.000125
Species_CULEX ERRATICUS	0.000000

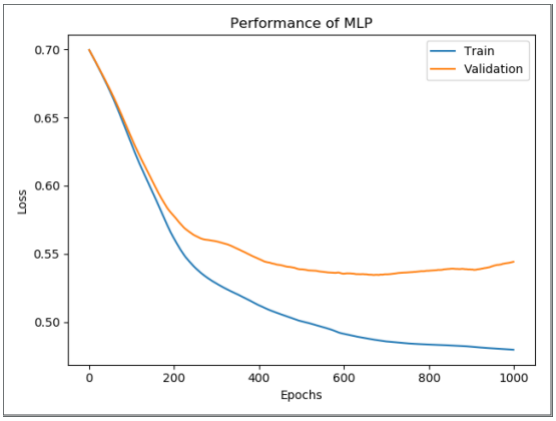
Feature importance from random forest model

Before training, we split the dataset into 70% training, 15% validation and 15% testing. Since the training set is imbalanced, we stratified the targets so that the training and validation set sampled the targets with respective to their observed ratios, five percent labeled one and 95 percent labeled zero. To help correct this imbalance we used the class weights parameter in the optimizer *Adam* with 9.5 to 1 ratio to train the dataset. To determine the learning rate, we trained with different ones, keeping track of the loss (error) as the model was training. If the model became stuck at an error that seemed “relatively” high, then we were considered this to be a local minimum; and likewise, we needed to adjust this parameter. If the error diverged, this meant that the learning rate was too large, and we missed the global minimum. Therefore, we ended up with a learning rate of 1e-3. Once we settled on the learning rate, we need to check the model for overfitting and, accordingly, the right amount of training (number of epochs).

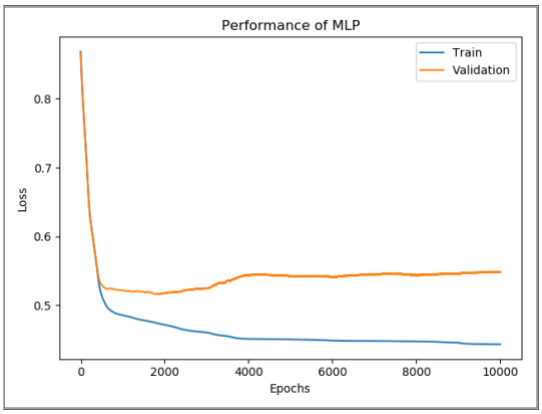
Using a full-batch (training the whole dataset at each iteration), we first trained the model for 10, 000 epochs, simultaneously feeding the model the validation set and calculating/storing the error. We then plotted the error of the training and validation sets, looking for a stable point at which the validation error increased monotonically (the point at which the model is overfitting). This occurred at 1,000 epochs. To prevent overfitting, we tried two options: early stoppage and L2-penalization (weight decay in *Adam*). The first option meant that we had to stop training at 1,000 epochs and the second option meant that we had to adjust *weight decay* to 0.005 to penalize large weights. After evaluating the performance of each method, we chose to stop the training early at 1,000 epochs.



No early stoppage or weight decay



Early Stoppage

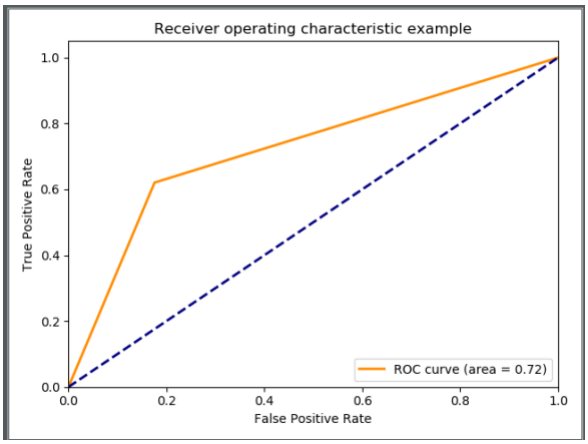


Weight_decay = 0.005

Results

Since the Kaggle challenge ranked submissions based on the area under the *Receiver Operator Curve* (auc), after training, we used the testing set to calculate the auc to evaluate our model's performance. The *Receiver Operator Curve* (roc) is a curve that plots the true positive rate against the false positive rate, and the auc is its definite integral from zero to 100. Out of a score

of 0.1, our model produced an auc of 0.72, but can be as low as 0.68, depending on how the weights are initialized.



Summary/Conclusion

In summary, multilayer perception is an effective machine learning tool, but there are many other mathematical and statistical models to consider. Some of challengers in the top ten on the leaderboard designed statistical/probabilistic model for this problem and did not choose a deep neural network or mlp. With more background in feature engineering, probability/statistics in data science (nonparametric statistics)—generalized additive models (GAM) is at the top of the leaderboard—and subject matter background (how does the weather related to some of the features), perhaps the results would be stronger or more suitable for such a problem. In the end, we showed that a deep mlp with intuitive functions and operations can provide result that are just as competitive to the more sophisticated algorithms out there.

References

- doyleax: https://github.com/doyleax/West-Nile-Virus-Prediction/blob/master/Final-NB.ipynb
- amir-jafari: https://github.com/amir-jafari/Deep-Learning/tree/master/Pytorch_/2-nn_module
- https://github.com/rhsu0268/West-Nile-Virus/blob/master/noaa_weather_qclcd_documentation.pdf (*weather.csv* metadata)
- Neural Network Design: http://hagan.okstate.edu/NNDesign.pdf