Hassles in medical appointment

Namrata Bilurkar, Suraj Nair
Northeastern University
360 Huntington Avenue
Boston, Massachusetts 02115

Abstract

The presented work illustrates the use of machine learning techniques, specifically Neural Networks, combined with use of classic optimization algorithms for prediction with patient data to predict if patients will show up or not for an appointment based on his medical conditions and/or previous history. We have used Tensor Flow library along with ProximalGradientDescentOptimizer algorithm to train the model. We also present the evaluation of other Optimizers for comparison. The controlled experimental results combined with our unique use of features showed that this new method with neural networks is significantly better than linear regression and Naive Bayes algorithm.

Introduction

Getting stood up is the worst. No one enjoys arriving at the restaurant just to receive a call from the supposed date who cannot make it because of a gold fish that needed veterinary care. But essentially, patient no-shows are the physicians equivalent to getting stood up. Except instead of simply wasting time, no-shows cost your practice money. Some estimate that patient no-shows cost individual physicians as much as $150,000 per year. Figures like this show why its essential to do all you can to reduce no-shows. But there are certain things you dont want to do when trying to decrease patient no-shows. Going about it the wrong way can waste time, or even worse, damage relationships with your patients. Quoted by Journal of Royal Society of Medicine 30% of patients miss their scheduled medical appointments. Why so?
A person makes an appointment with a doctor, receives all the instructions and does not show up. Who is to be blamed? The inspiration behind this project is simple, what if it is possible to predict if a person will show up for an appointment? There could be prodigious advantages of this prediction, namely proper utilization of health care resources.

Background

Terminology

S.No	Term	Explanation
1	Training set	Contains a pair of input and expected output, one to one mapping
2	Test Data	Data used to verify that a given set of input produces some expected result
3	Tensor Flow	An open-source software library for Machine Intelligence
4	Optimizers	The Optimizer base class in Tensor Flow provides methods to compute gradients for a loss and apply gradients to variables. A collection of subclasses implement classic optimization algorithms

Data Preprocessing

Data goes through a series of steps during preprocessing:

- Data Cleaning: Data is cleansed through processes such as filling in missing values, smoothing the noisy data, or resolving the inconsistencies in the data.
- Data Integration: Data with different representations are put together and conflicts within the data are resolved.
- Data Transformation: Data is normalized, aggregated and generalized.
- Data Reduction: This step aims to present a reduced representation of the data in a data warehouse.
- Data Discretization: Involves the reduction of a number of values of a continuous attribute by dividing the range of attribute intervals.

Related Works

Previously, for the same data set machine learning techniques like Bayesian Analysis and Linear Regression gave

a prediction accuracy of 69%.

Project Methodology

The first step here is to perform pre-processing steps on the data to prepare it for the training. This includes removal of any kind of noises and irregularities in data.

1. **Data Cleaning:**

   (a) It is immediately apparent that some of the column names have spelling errors, so clearing the errors here would be the first task as these column names should be meaningful.

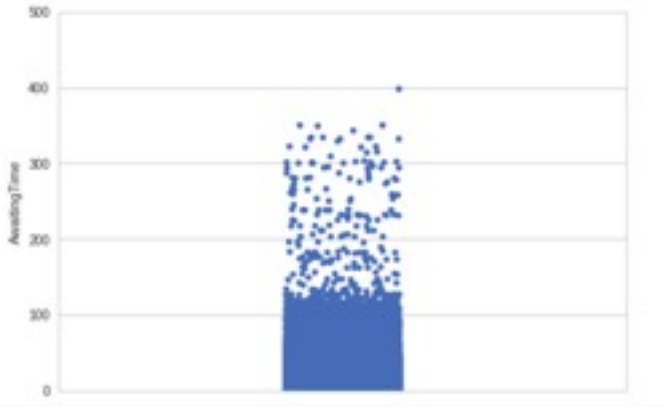
   (b) We check for any erroneous values and inputs that are not numbers in data. We do this by performing:

```
print('Age:',sorted(noShow.Age.unique()))
print('Gender:',noShow.Gender.unique())
print('DayOfTheWeek:',noShow.DayOfTheWeek.unique())
print('Status:',noShow.Status.unique())
print('Diabetes:',noShow.Diabetes.unique())
print('Alchoholism:',noShow.Alchoholism.unique())
print('Hypertension:',noShow.Hypertension.unique())
print('Handicap:',noShow.Handicap.unique())
print('Smokes:',noShow.Smokes.unique())
print('Scholarship:',noShow.Scholarship.unique())
print('Tuberculosis:',noShow.Tuberculosis.unique())
print('Sms_Reminder:',noShow.Sms_Reminder.unique())
print('AwaitingTime:',
sorted(noShow.AwaitingTime.unique()))
print('HourOfTheDay:',
sorted(noShow.HourOfTheDay.unique()))
```

   It is clear that we do not have any NaNs anywhere in the data. However, we do have some impossible ages such as -2 and -1, and some unrealistic ages such as 100 and beyond. It is totally possible to live 113 years and celebrate living so long, and some people do live that long, but most people don't. So, we are treating the ages greater than 95 as outliers.

   (c) Outliers in Awaiting Time:

```
sns.stripplot(data = noShow,
              y = 'AwaitingTime',
              jitter = True)
sns.plt.ylim(0, 500)
sns.plt.show()
```



Clearly, the data starts to thin out after AwaitingTime of 150 days. There is one observation at 398 days, which is likely an outlier. There are almost no observations beyond 350 days, so we remove anything beyond 350 days which will include that observation entry of 398 day too.

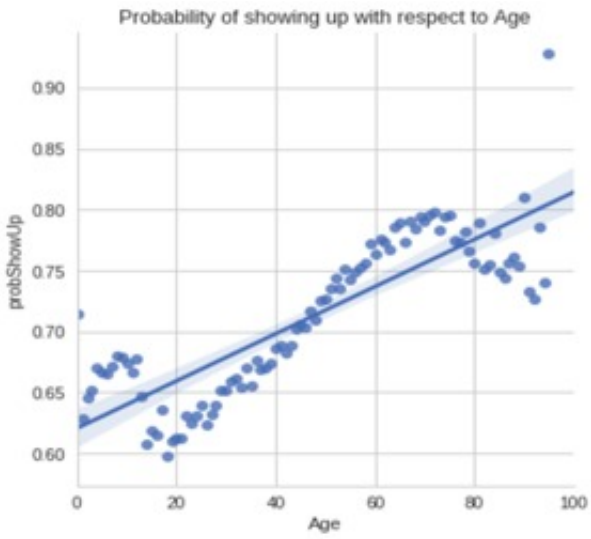
2. **Data Spliting:**

   We split the data into training and testing data. After a lot of trial and error, we have found that about 297500 samples in the training data are most helpful to our classifier to properly get the best fit. Beyond that it overfits and below that it underfits.

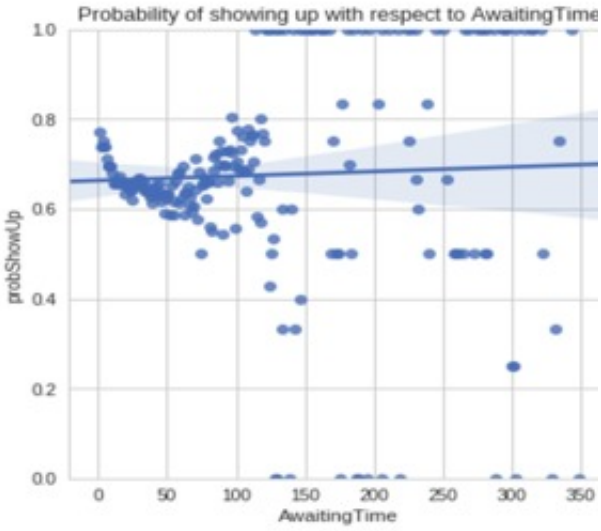
3. **Feature Selection:**

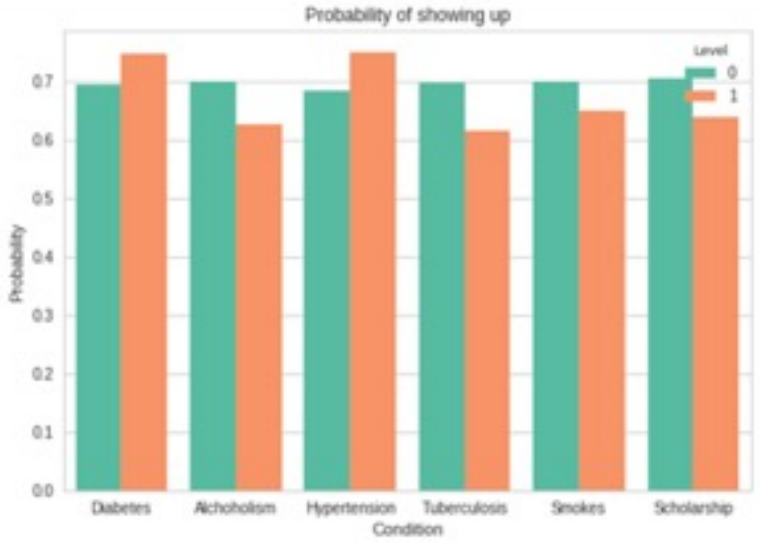
   Analyzing the probability of showing up with respect to different features

   (a) *Age:*

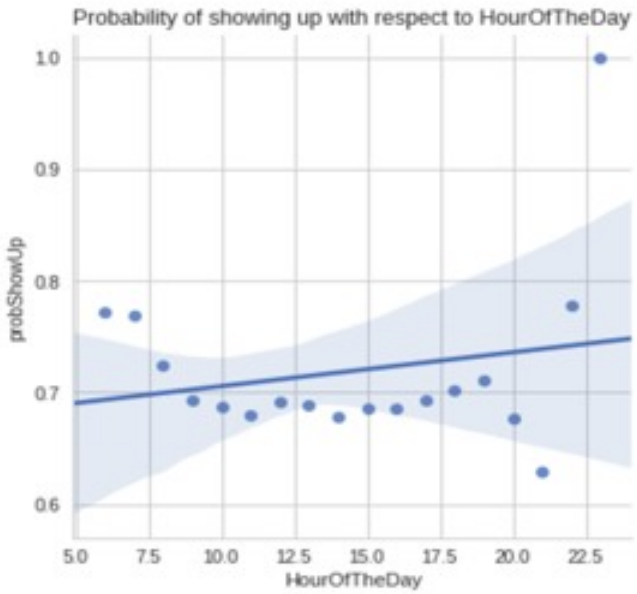


   (b) *Awaiting Time:* It is the rounded number of days from registration to appointment.





(c) *Hour of the Day:* We created a new feature called HourOfTheDay, which will indicate the hour of the day at which the appointment was booked.

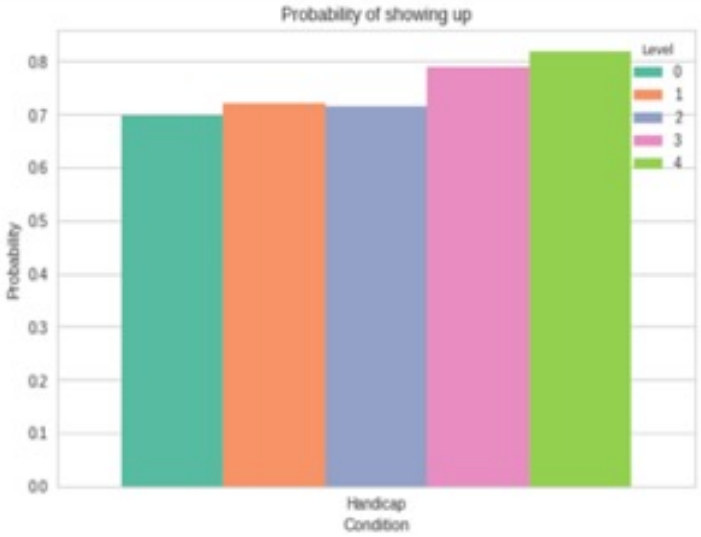


Clearly, HourOfTheDay and AwaitingTime are not good predictors of Status, since the probability of showing up depends feebly on the HourOfTheDay and not at all on the AwaitingTime. The significantly stronger dependency is observed with respect to Age.

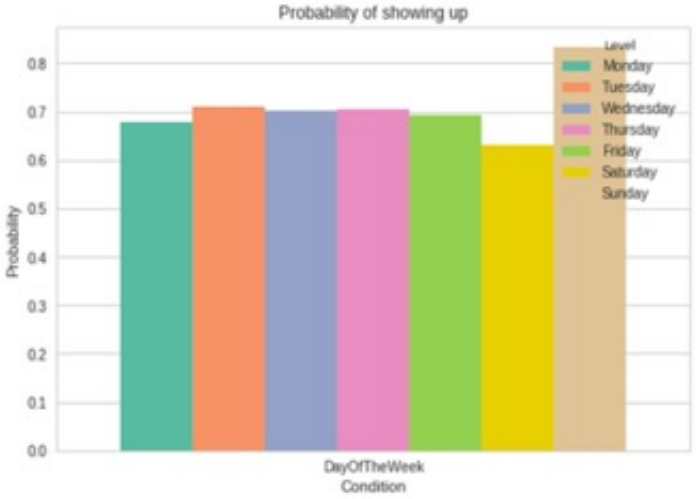
(d) *Conditions:*

(e) *Gender:*

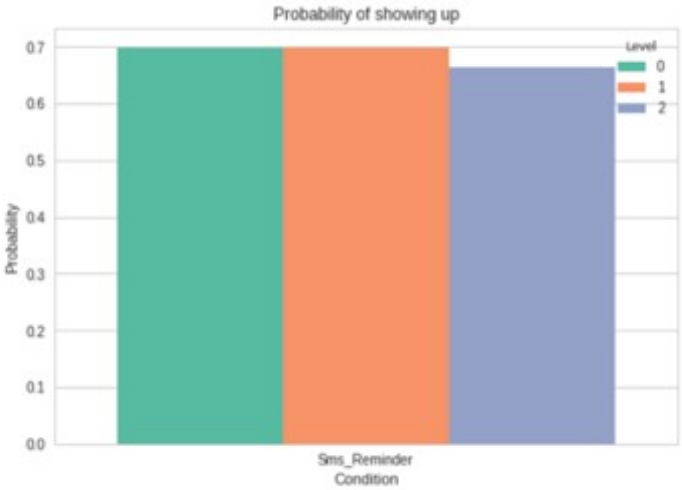
(f) *Handicap Condition:*



(g) *Day of the week:*



(h) *SMS reminder condition:*



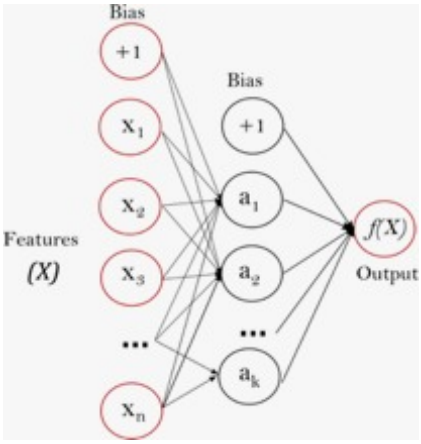
We are not considering SMS Reminder as a cause because it is not a cause for not showing up, rather it is a feature of interest that is unrelated with showing/not-showing to the appointment.

4. **Implementation:**

(a) **Multi-layer Perceptron (MLP)** is a supervised learning algorithm that learns a function

$$f(\cdot) : R^m \rightarrow R^o$$

by training on a dataset, where m is the number of dimensions for input and o is the number of dimensions for output. Given a set of features $X = x_1, x_2, ..., x_m$ and a target y , it can learn a non-linear function approximator for either classification or regression. It is different from logistic regression, in that between the input and the output layer, there can be one or more non-linear layers, called hidden layers. Figure below shows an MLP having one hidden layer with scalar output.



The leftmost layer, known as the input layer, consists of a set of neurons $\{x_i|x_1, x_2, ..., x_m\}$ representing the input features. Each neuron in the hidden layer transforms the values from the previous layer with a weighted linear summation $w_1x_1 + w_2x_2 + .. + w_mx_m$, followed by a non-linear activation function

$$g(\cdot) : R \rightarrow R_-$$

like the hyperbolic tan function. The output layer receives the values from the last hidden layer and transforms them into output values.
The module contains the public attributes $coefs_-$ and $intercepts_-$. $coefs_-$ is a list of weight matrices, where weight matrix at index i represents the weights between layer i and $i + 1$. $intercepts_-$ is a list of bias vectors, where the vector at index i represents the bias values added to layer $i + 1$.

The advantages of Multi-layer Perceptron are:

- Capability to learn non-linear models.

- Capability to learn models in real-time (on-line learning) using $partial\_fit$.

The disadvantages of Multi-layer Perceptron (MLP) include:

- MLP with hidden layers have a non-convex loss function where there exists more than one local minimum. Therefore different random weight initializations can lead to different validation accuracy.
- MLP requires tuning a number of hyperparameters such as the number of hidden neurons, layers, and iterations.
- MLP is sensitive to feature scaling.

Class **MLPClassifier** implements a multi-layer perceptron (MLP) algorithm that trains using **Backpropagation**.

(b) **Tensor Flow:** TensorFlow is an open source software library for machine learning across a range of tasks, and developed by Google to meet their needs for systems capable of building and training neural networks to detect and decipher patterns and correlations, analogous to the learning and reasoning which humans use. To train the data, we have several Optimizers to use.

The Optimizer base class provides methods to compute gradients for a loss and apply gradients to variables. A collection of subclasses implement classic optimization algorithms such as GradientDescent and Adagrad.

Optimizer class by itself should never be instantiated, but instead one of the subclasses has to be instantiated before using it.

```
tf.train.Optimizer
tf.train.GradientDescentOptimizer
tf.train.AdadeltaOptimizer
tf.train.AdagradOptimizer
tf.train.AdagradDAOptimizer
tf.train.MomentumOptimizer
tf.train.AdamOptimizer
tf.train.FtrlOptimizer
tf.train.ProximalGradientDescentOptimizer
tf.train.ProximalAdagradOptimizer
tf.train.RMSPropOptimizer
```

Experiments

The values in each iteration is the accuracy value of the trained data.

Optimizer	Iterations				Average
	0	1	2	3	iteration
Gradient Descent Optimizer	0.99996	1.00000	0.99998	0.99998	0.999899
Ada Delta Optimizer	0.24222	0.72326	0.30546	0.54856	0.489486
AdaGrad Optimizer	0.53552	0.30808	0.27428	0.71694	0.466553
Adam Optimizer	0.69748	0.72012	0.32234	0.35820	0.496193
Ftrl* Optimizer	0.66778	0.61024	0.41806	0.66670	0.620326
Proximal Gradient Descent Optimizer	0.99998	1.00000	0.99980	0.99992	0.999896
Proximal Adagrad Optimizer	0.66106	0.67908	0.75826	0.32708	0.572383
RMSProp Optimizer	0.68934	0.62264	0.28478	0.30160	0.452583

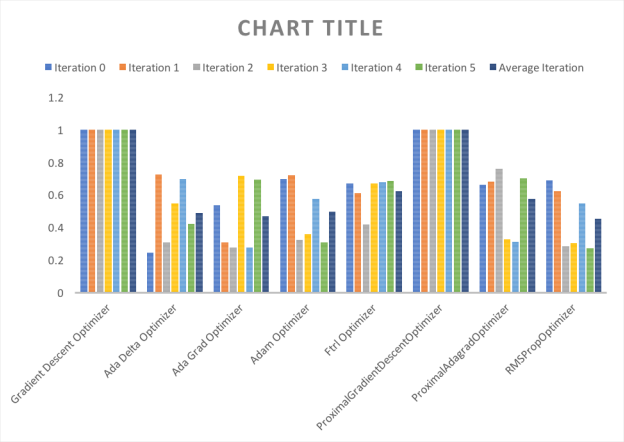
*Ftrl optimizer uses Follow-the-regularized-leader Proximal algorithm.

A rectifier is an activation function defined by:

$$f(x) = max(0, x)$$

where x is the input to a neuron. A unit employing the rectifier is also called a **rectified linear unit (ReLU)**. In mathematics, the softmax function, or normalized exponential function, is a generalization of the logistic function that "squashes" a K-dimensional vector of arbitrary real values to a K-dimensional vector of real values in the range (0, 1) that add up to 1.

The chart below shows the results with 5 iterations of each optimiser:



Conclusion

The calculations shown have the greatest individual accuracy, indicating that it contributes the most to the accuracy. However, in all cases adding more relevant features improved the accuracy. On its own, the algorithms gave us not so accurate results, indicating that perhaps there is a better way of estimating the problem statement. The application achieved close to 73% accuracy by using a neural network with Back propagation for learning. Similarly, training data with Optimizers in tensor flow library gave us an accuracy close to 99%. We have shown the comparison of using different optimizer algorithms where accuracy reduces when not trained properly. This proves that the data input provided isnt an overfit. Thus, we conclude that the neural network model with Tensor flow library could successfully predict if the patient will show up for the appointment or not.

References

1. Martn Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng of Google Research on **TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems**
2. Github repository of the TensorFlow library
3. Joerg Evermann of Memorial University of Newfoundland, Jana-Rebecca Rehse and Peter Fettke of German Research Center for Artificial Intelligence and Saarland University on **XES Tensorflow Process Prediction using the Tensorflow Deep-Learning Framework**
4. Back propagation with Neural Networks
5. Dataset for this project has been used from Kaggle