Northeastern University

Predicting Parkinson's Disease Progression using Regression analysis

IE 7374

Machine Learning in Engineering

Group-20

NUID	Team Members	Email
001305283	Abhilash Hemaraj	hemaraj.a@northeastern.edu
001348315	Ayush Venkatesh	venkatesh.ay@northeastern.edu
001097953	Porom Sivam Kalita	kalita.p@northeastern.edu
001052134	Sanghamitra Shanmugam	shanmugam.sa@northeastern.edu

Abstract

Parkinson’s is an illness that affects the central nervous system. It is a chronic degenerative movement disorder that is caused by the brain’s lack of ability to produce dopamine, which in turn leads to the deterioration of motor functions. Parkinson’s worsens with time and it is important to track the progression to mitigate symptoms. The progression can be quantified using the Unified Parkinson’s Disease Rating Scale (UPDRS) for remote monitoring using telemonitoring frameworks. This project focuses on using Parkinson’s Telemonitoring dataset, a set of voice measurements as a biomarker, to predict UPDRS scores. We implement linear and non-linear models from scratch and report on the regression analysis.

Introduction

This dataset is composed of a range of biomedical voice measurements with early Parkinson’s disease of around 42 people. They were recruited to a 6 month trial of a telemonitoring device for the purpose of remote symptom progression monitoring. The recordings were automatically captured in the patient’s home. The main aim of this project is to predict the motor UPDRS score (Unified Parkinson’s Disease Rating Scale) which is linearly interpolated. The dataset we have are clinically useful features of average Parkinson's Disease progression which were extracted by characterizing speech with signal processing algorithms. Even though medication and surgical intervention can prolong or hold back the progression of this disease and alleviate some of the symptoms, there is no available cure. Hence, early diagnosis is critical in order to improve the patient’s quality of life.

The motor_UPDRS ranges from 0-108, with 0 denoting symptom-free and 108 severe motor impairment, and encompasses tasks such as speech, facial expression, tremor, and rigidity.

Dataset description:

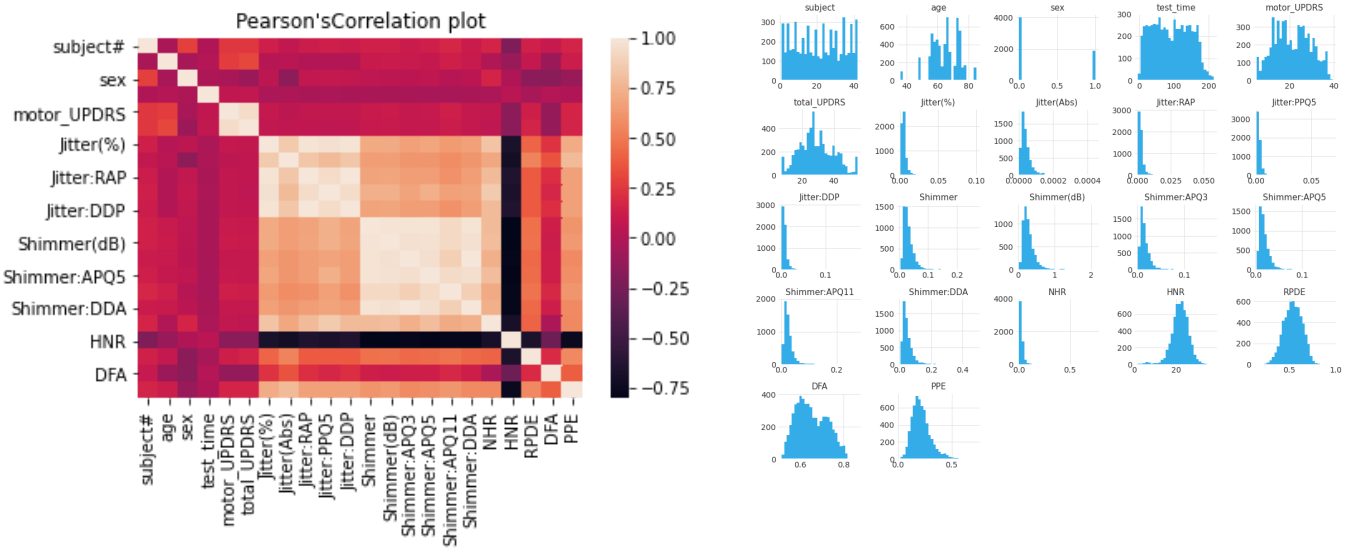
The data is in CSV format. The rows of the CSV file contain an instance corresponding to one voice recording. There are around 200 recordings per patient, the subject number of the patient is identified in the first column

- subject - Integer that uniquely identifies each subject
- age - Subject age
- sex - Subject gender '0' - male, '1' - female
- test_time - Time since recruitment into the trial. The integer part is the number of days since recruitment.
- motor_UPDRS - Clinician's motor UPDRS score, linearly interpolated
- total_UPDRS - Clinician's total UPDRS score, linearly interpolated
- Jitter(%), Jitter(Abs), Jitter: RAP, Jitter: PPQ5, Jitter: DDP - Several measures of variation in fundamental frequency
- Shimmer, Shimmer(dB), Shimmer: APQ3, Shimmer: APQ5, Shimmer: APQ11, Shimmer: DDA - Several measures of variation in amplitude
- NHR, HNR - Two measures of the ratio of noise to tonal components in the voice
- RPDE - A nonlinear dynamical complexity measure
- DFA - Signal fractal scaling exponent
- PPE - A nonlinear measure of fundamental frequency variation

Methods (description of models we have used)

1. **Linear Regression with Regularization**: Use a high bias, simpler model to build a baseline for our analysis.
2. **K-nearest neighbors regressor**: A non-parametric model that uses a distance metric like Euclidean distances for a lazy learning objective.
3. **Decision Trees**: A non-parametric supervised learning model for regression. The goal is to predict target value by learning simple decision rules.
4. **Neural Networks**: Use Multi-layer perceptrons and Backpropagation for non-linear modeling of data.

Explanatory data analysis (EDA) - including feature engineering, feature selection, statistical analysis, ...



The left plot in the above is a correlation plot of all the features. We can see that there is a lot of correlation between some of the variables. The total_updrs variable is very similar to the motor_updrs variable, hence we remove total_updrs. However, there is a lot of correlation between many of the variables, so we removed variables that were correlated beyond a certain threshold (0.95), and then observed the effect on our models. The effect was not substantial, hence we don't include that in our report.

The right plot in the above is a density plot of all the features. We can see that many of the features are not normally distributed, hence we used the min-max scaler to normalize our data before applying our models.

Results (full description of your model and the results)
Decision Trees
Decision trees build regression models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes.

We decide the split based on calculating the variance reduction for each split. We will loop through all features by traversing through all possible values of the dataset. In the context of regression, we use variance as a measure of impurity. Initially, we will compute the variance for the root and also for the divided datasets. Then we will compute the variance reduction by the below formula:

$$\textbf{Variance\_reduction} = \textbf{var(parent)} - \sum \mathbf{w_i}\ \textbf{var(child)}$$

Where w_i is the relative size of the child with respect to the parent

Implementation Description:
We have two classes which are as follows:

- Node: There exist two kinds of nodes - Decision nodes and leaf nodes. The decision nodes will have feature_index, left, right, variance_reduction variables associated with them whereas the leaf node will only have value variables.
- Decision Tree Regressor
    - **build_tree**() : This is a recursive function. We check whether the stopping criteria are met (minimum samples split and the maximum depth). If they are not met, we find the best split for the current dataset(using get_best split()). If the variance reduction by the best split is greater than 0, we split the dataset and we recursively call the build tree function for left and right subtrees. Finally, we create the decision node after recursively building the tree. Once we have reached the bottom of the tree, we will calculate the leaf value (using calculate_leaf_value())
    - **get_best_split**(): We will loop through all the features by traversing through all possible values in the dataset. After splitting based on each threshold, we will compute the reduction

in an impurity by calculating the variance reduction. We compare the information gained and decide on the best split.

We randomly selected the minimum samples per split to 6 and maximum depth to also 6 while running the decision tree regressor. We fit the training data and printed the tree. Our model resulted in a rmse of 2.320050

Linear Regression

Linear Regression is a linear methodology of modeling data to predict a continuous variable. It is a model of high bias and is often the first choice to build a baseline for your analysis. In this model, we are trying to find the best coefficients for the variables that give us the best fit. A typical linear regression model with a single variable X and a target variable y will look like the following:

Y = Intercept + Coefficient * X

Ordinary Least Squares:

By minimizing the sum of squared errors this model forms a closed-form solution to finding the parameters.

$$\hat{\boldsymbol{\beta}} = (\mathrm{X}^\mathsf{T}\mathrm{X})^{-1}\mathrm{X}^\mathsf{T}\mathbf{y}. \qquad w = (\mathrm{X}^T\,\mathrm{X} + \lambda n\,\mathrm{I})^{-1}\,\mathrm{X}^T\,y$$

Without regularization[1] With regularization[1]

Gradient Descent and Stochastic Gradient Descent:

The Family of Gradient Descent regressors uses an iterative process to update the weights/parameters of the model. The update gradient is the product of learning-rate and cost-gradient. In linear regression, the cost-gradient is based on the sum of squared residuals.

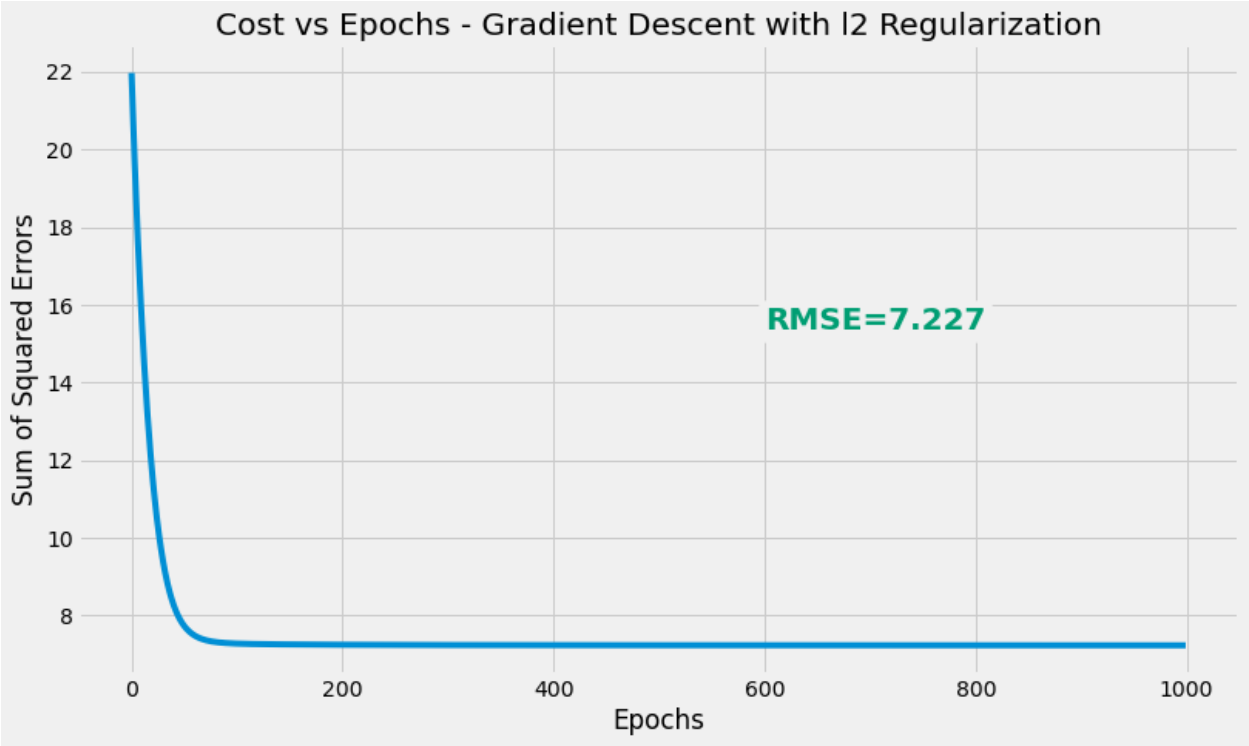
Weights = weights - learning-rate * cost-gradient

With regularization[1]:

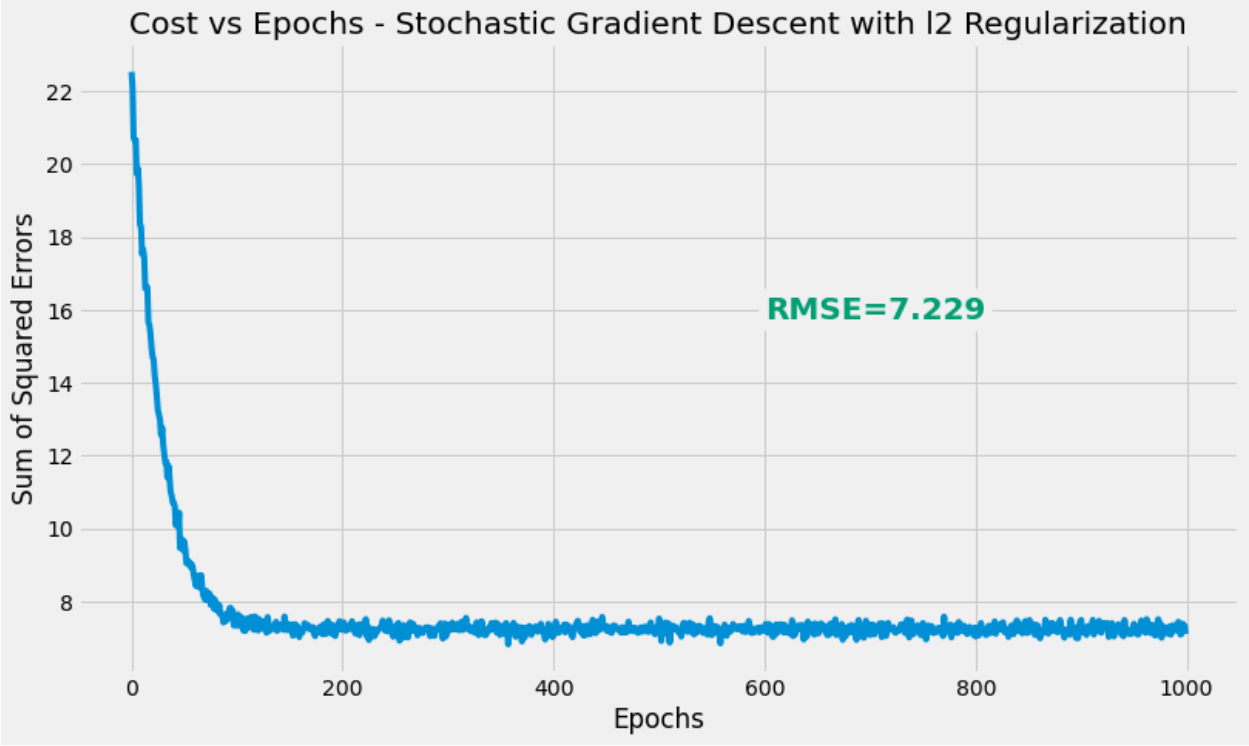
$$\min_{\beta \in \mathbb{R}^p} \left\{ \frac{1}{N} \|y - X\beta\|_2^2 + \lambda \|\beta\|_1 \right\}$$

We have a single master class for Linear Regression that fits three different types of linear models viz Ordinary Least Squares, Gradient Descent, and Stochastic Gradient Descent.

Gradient Descent Regressor Loss Curve:



Stochastic Gradient Descent Regressor Loss Curve:



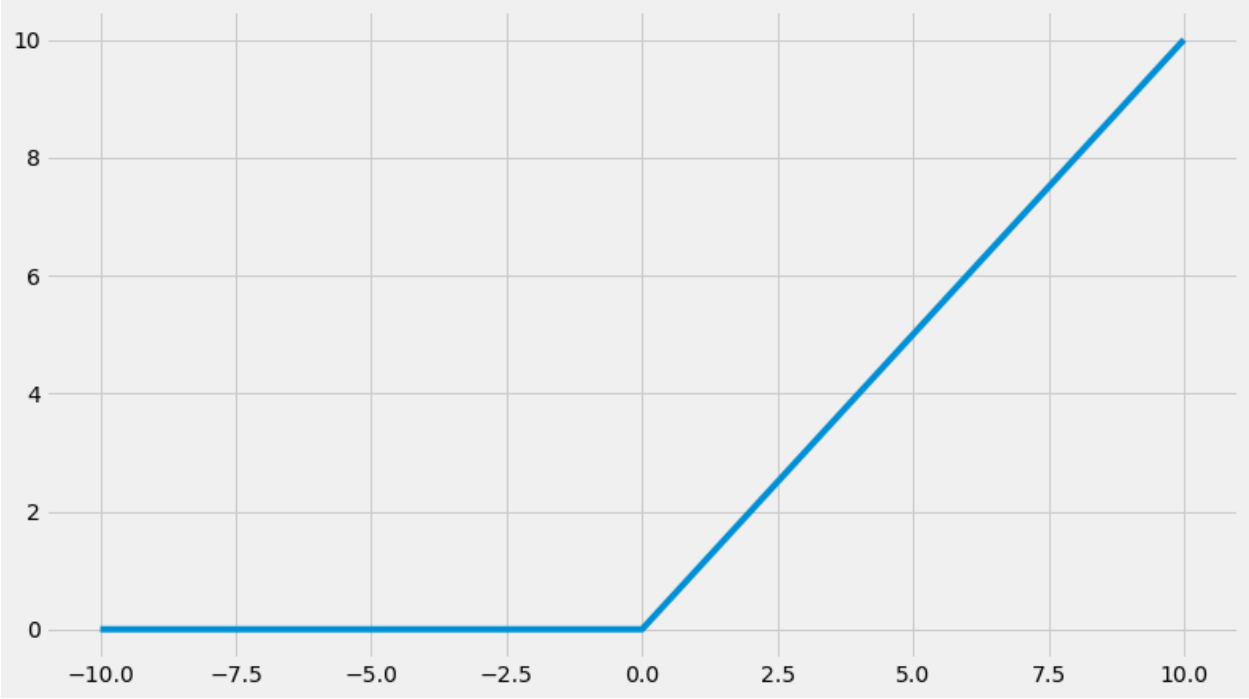
Neural Networks

Neural Networks form the foundation for Deep Learning, wherein the idea is to use perceptrons that roughly mimic information passing through neurons of the brain

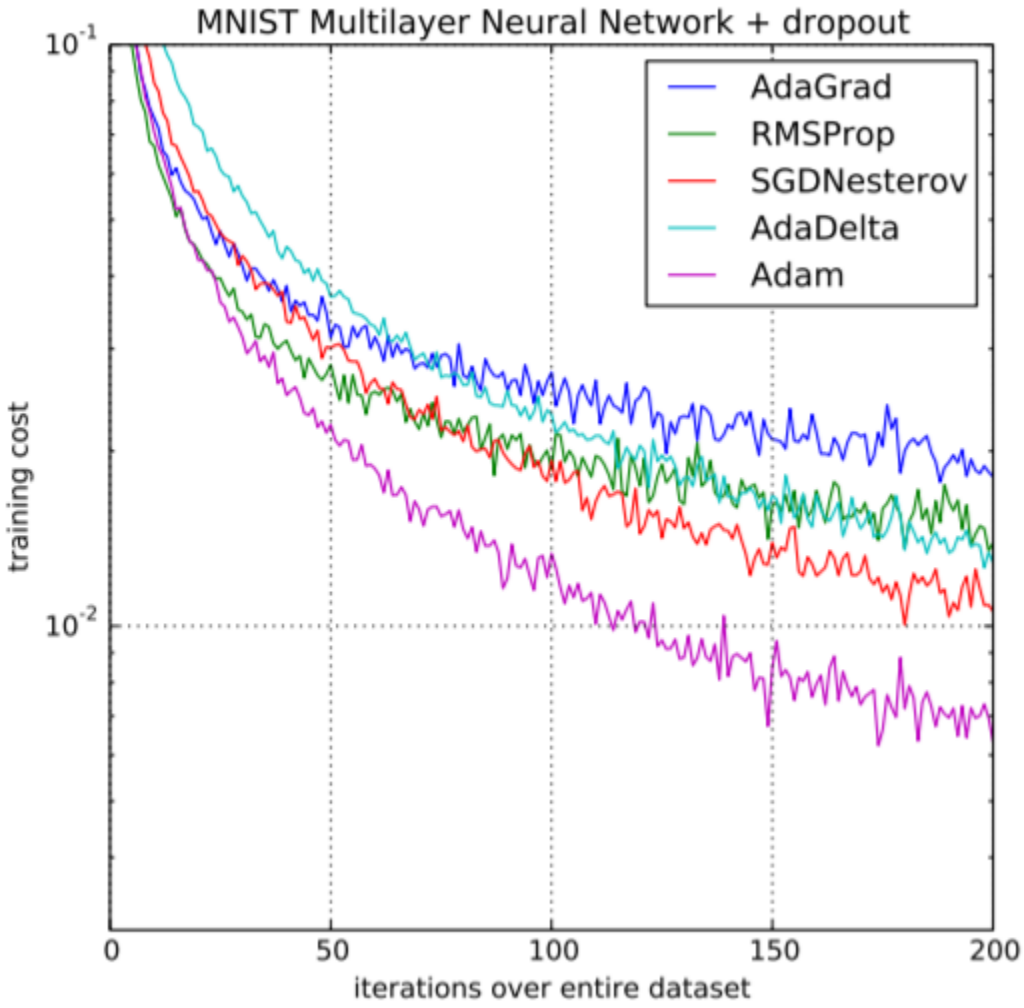
For this particular use case, we have modeled the Multi-Layer Perceptron model. MLP's help in modeling the non-linear relationships between the predictor and the target variable and usually leads to strong models. But neural networks are stochastic in nature and have a tendency to not converge. They are computationally expensive and it takes a lot of effort to find the right hyperparameters.

We are using the ReLU activation function at every hidden layer and keeping the output linear. For optimization purposes, we are using the adam optimizer and backward propagation to update the weights and biases.

Rectified Linear Unit Activation Function:



Adaptive Moment Estimation:



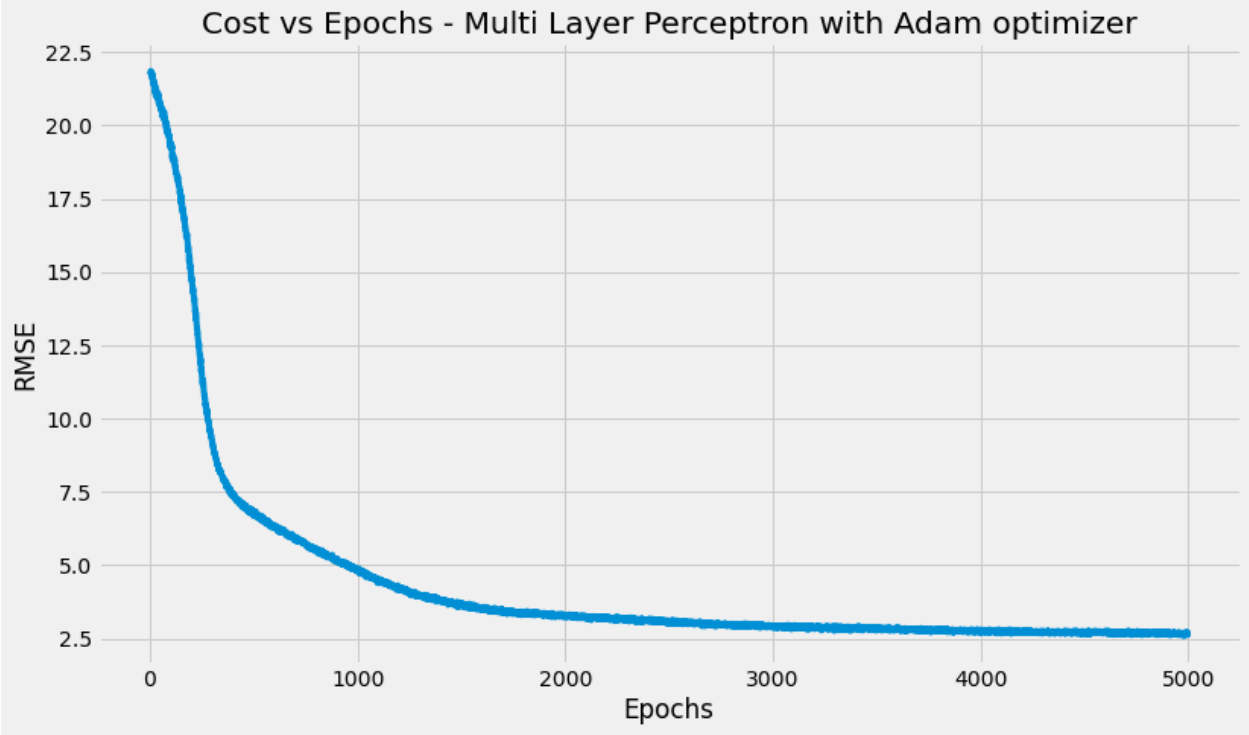
[2] Adam: Cost comparison with other optimizers on MNIST task

Adam computes adaptive learning rates for its hyperparameters from moments of the gradients. It is a combination of RMSProp and AdaGrad algorithm and has been found suitable for most deep learning tasks.

Our implementation of Neural Networks uses the TensorFlow optimizer and gradient tape functions to keep track of and update the gradients. In our experiments, we found that step had a great effect on performance when compared to backpropagation done from scratch.

Our NNregressor class consists of a forward_propagation to perform the feed-forward operation using input data and weights and biases. It contains the backward_propagtion function to compute loss and update the parameters on the way backward.

After ample amounts of experimentation, we found that for a learning rate of 0.07, and with the adam optimizer the four-layer neural network, with sizes [20, 15, 15, 1] was able to achieve a test rmse of 2.85.



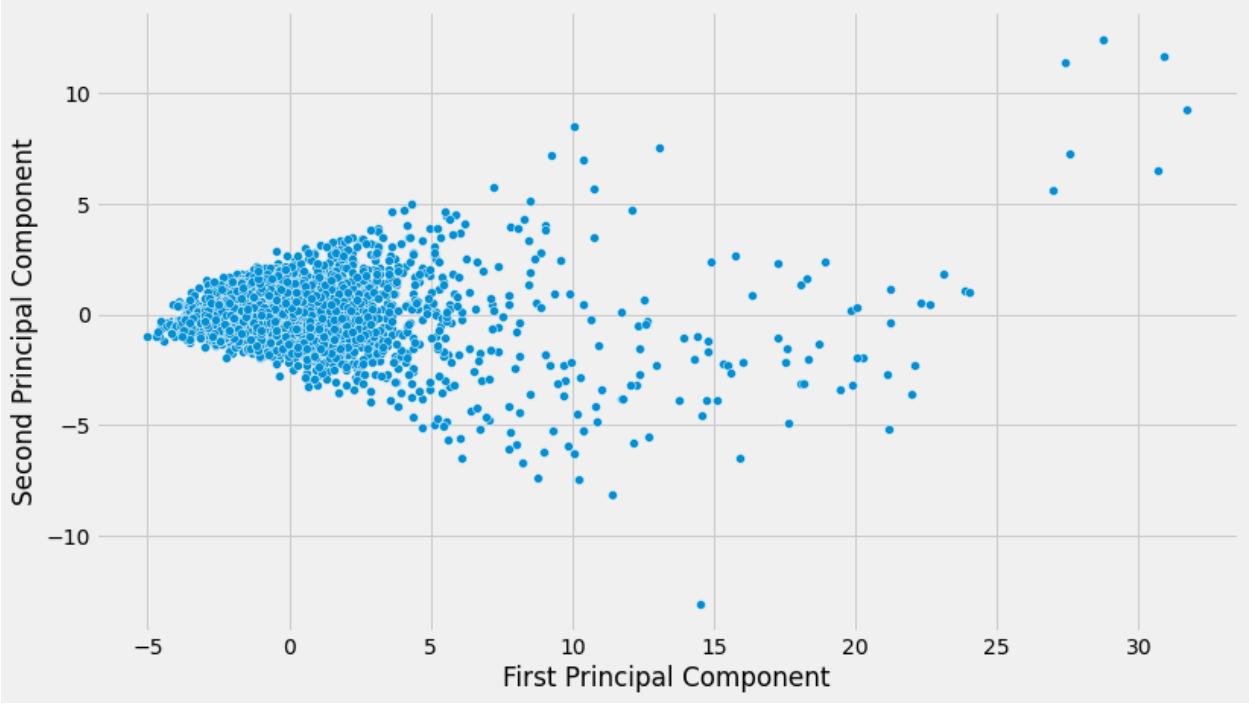
Principal Component Analysis

Principal Component Analysis or PCA is a popular Dimensionality reduction technique that transforms a set of variables and projects them into lower dimensions without losing much information.

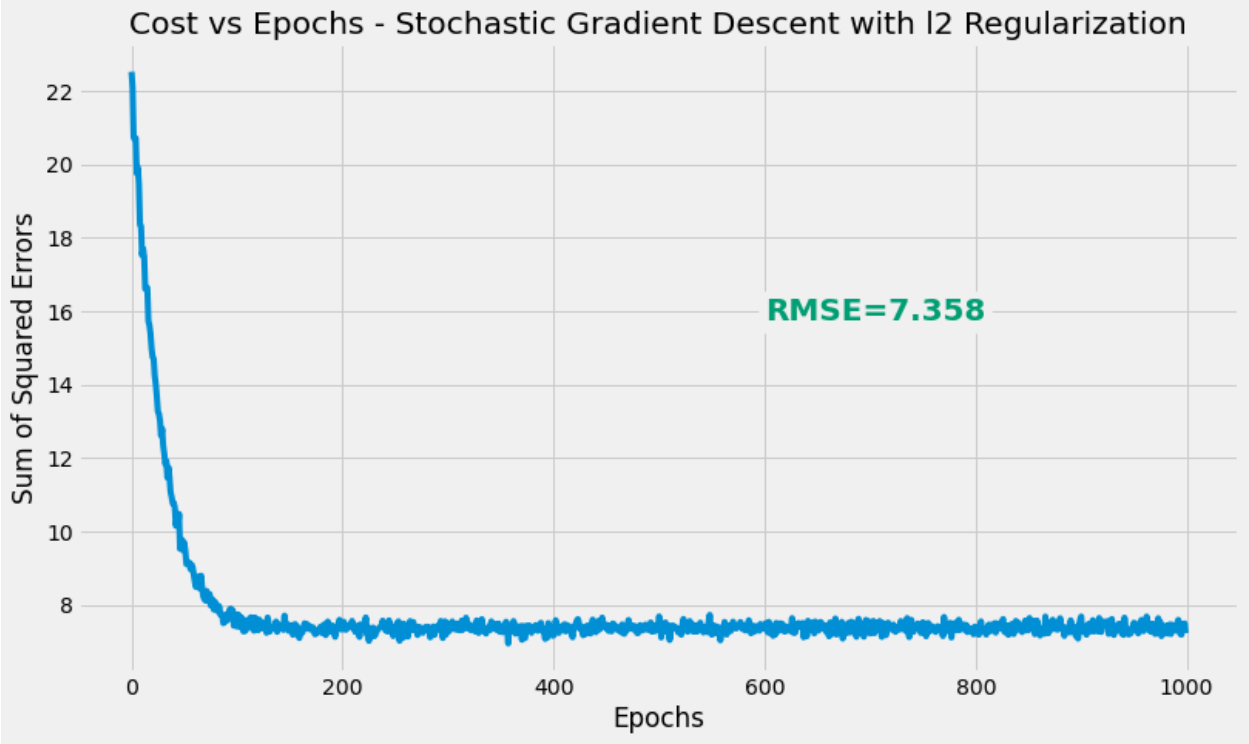
PCA Algorithm:

1. Mean normalize the variables: X- u
2. Compute covariance of X
3. Compute eigenvalues and eigenvectors of the covariance matrix.
4. Keep the top components based on the magnitude of eigenvalues.

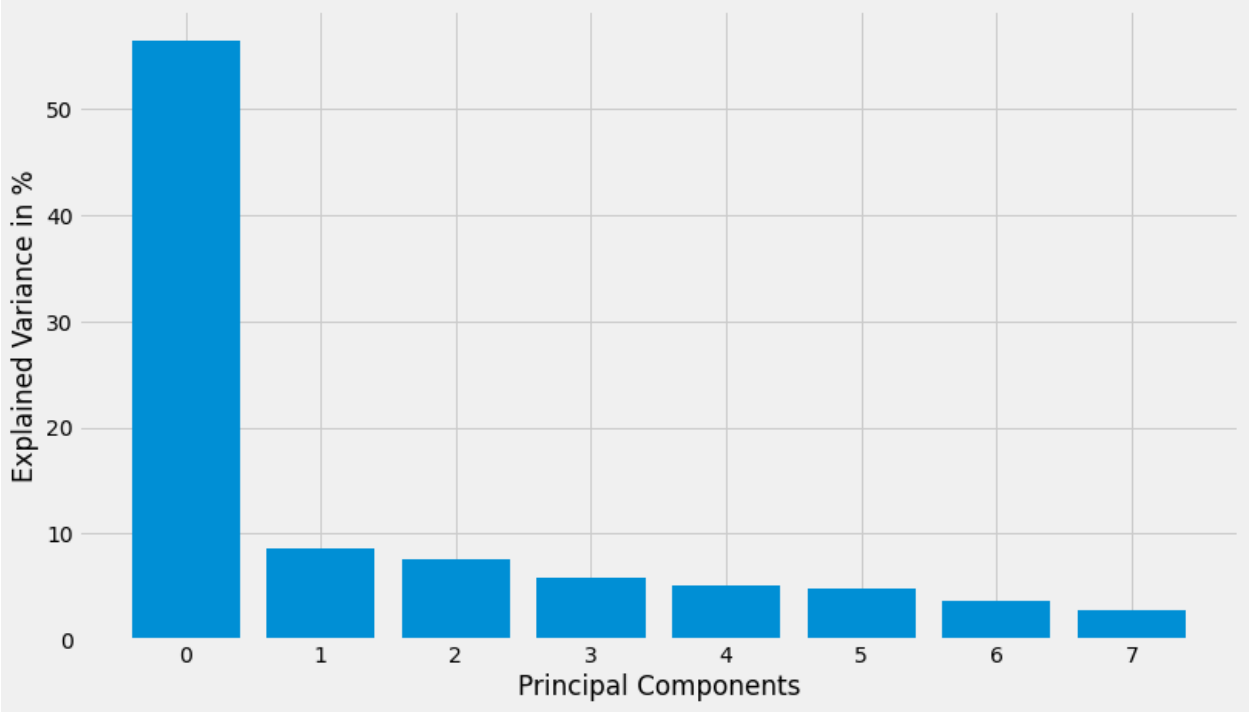
Since there are 20 variables as predictors, we wanted to try and reduce dimensionality to see if it made a significant difference in modeling:



By visualizing the first two components we can understand the spread of the components.



Above is the linear regression model fit on the top 8 Principal Components, with the below explained variances:



As it is evident from the above plot that the first 8 components explain a

total of 95 % variance combined. But that didn't translate to a higher rmse score in linear regression.

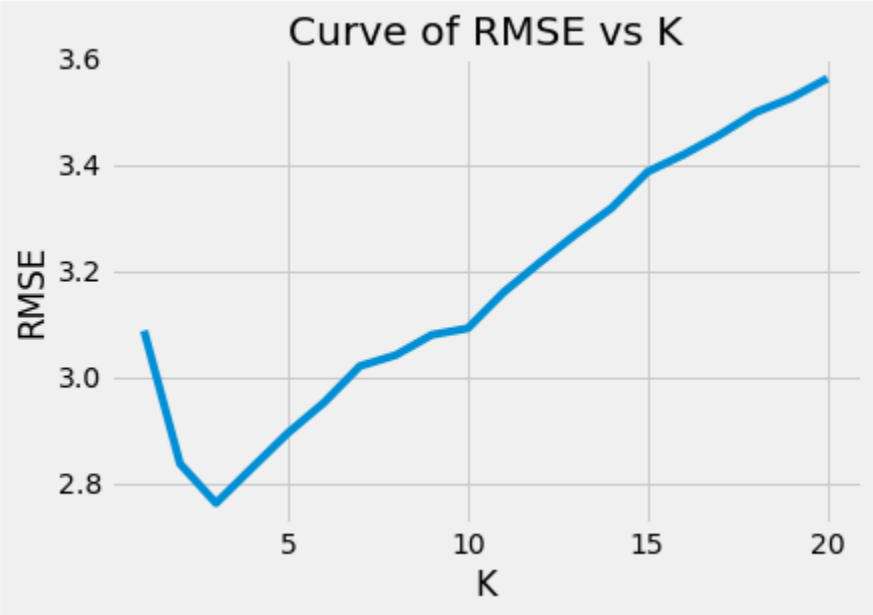
K-Nearest-Neighbours

K-Nearest-Neighbours or KNN is a non-parametric algorithm developed in 1951. It can be used for classification and regression. Given a new data point, the algorithm predicts its output by first selecting the k closest neighbors. Then, depending on whether our task is classification or regression, we predict in one of two ways:

1) Classification: We observe the majority class among the k nearest neighbors and set that as our output class.
2) Regression: We calculate the average values of the output labels of the k neighbors and assign that as our output.

One immediate question that arises. How to choose the value of k? The way we did it is by arbitrarily running the algorithm for values of k from 1 to 20. We observe that beyond a certain point the value of k keeps increasing hence, we choose k such that our root mean squared error (RMSE) is the lowest.

Here is a plot of the RMSE and K for our dataset:



As we can see in the above graph, at k = 4, we have the lowest value for RMSE. Beyond this, the RMSE increases continuously. Hence we choose k=4 as the optimal value of k.

Comparisons with Pre-built Libraries

We used the Pycaret module to fit almost all available regression-based models. Below is an aggregation of the comparison:

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
et	Extra Trees Regressor	0.1541	0.0721	0.2665	0.9989	0.0138	0.0083	0.3370
rf	Random Forest Regressor	0.2109	0.1601	0.3982	0.9976	0.0189	0.0105	0.8440
dt	Decision Tree Regressor	0.1331	0.2013	0.4424	0.9970	0.0184	0.0062	0.0190
lightgbm	Light Gradient Boosting Machine	0.4505	0.4178	0.6449	0.9937	0.0357	0.0253	0.2690
gbr	Gradient Boosting Regressor	1.5349	4.2885	2.0679	0.9351	0.1125	0.0887	0.5280
knn	K Neighbors Regressor	2.8111	17.9371	4.2319	0.7292	0.2151	0.1576	0.0130
ada	AdaBoost Regressor	4.2227	24.4052	4.9379	0.6314	0.2638	0.2553	0.1340
lr	Linear Regression	5.9977	51.5184	7.1757	0.2228	0.3676	0.3656	0.0210
lar	Least Angle Regression	5.9986	51.5201	7.1759	0.2228	0.3677	0.3657	0.0080
br	Bayesian Ridge	6.0427	52.3742	7.2355	0.2100	0.3702	0.3690	0.0080
ridge	Ridge Regression	6.0452	52.5077	7.2448	0.2080	0.3708	0.3697	0.3610
huber	Huber Regressor	6.0522	54.0386	7.3489	0.1849	0.3705	0.3606	0.0390
en	Elastic Net	6.2474	55.8550	7.4727	0.1574	0.3861	0.3875	0.0060
lasso	Lasso Regression	6.2540	55.8992	7.4757	0.1568	0.3865	0.3885	0.3390
omp	Orthogonal Matching Pursuit	6.7163	61.1283	7.8167	0.0783	0.4014	0.4174	0.0050
llar	Lasso Least Angle Regression	6.9792	66.5459	8.1563	-0.0034	0.4200	0.4416	0.0050
par	Passive Aggressive Regressor	9.3504	137.8130	11.2263	-1.0784	0.6988	0.4931	0.0100

From the above table, we can say that generally, tree-based models are performing well on the telemonitoring dataset. This holds true in our hand-implemented models as well, where Decision Trees performed the best out of the models.

Discussion

We explored the measure of motor UPDRS using various regression methods. Using the Pycaret library we could generate comparisons on hand-built models to prebuilt ones. Looking at the above figure it is evident that Tree-based models perform really well on the task. This is true in our experiments as well since our implementation of Decision Trees performed really well with a rmse of 2.32.

References

[1] https://en.wikipedia.org/wiki/Linear_regression

[2] Kingma, D.P. and Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[3]https://towardsdatascience.com/math-neural-network-from-scratch-in-python-d6da9f29ce65

[4] https://archive.ics.uci.edu/ml/datasets/parkinsons+telemonitoring