

Predicting the Azimuth Angle of an HRIR

October 9, 2022

1 Introduction

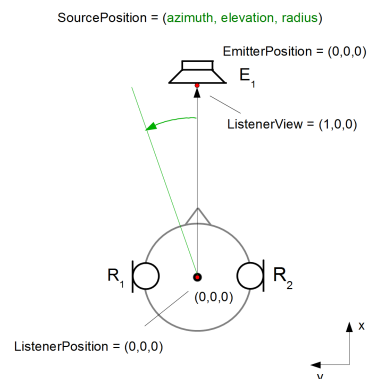
In human hearing, sound is localized via the sound signals that arrive to the right and left ear, called a binaural signal. In practice, a binaural signal can be digitally constructed with an arbitrary sound and a set of laboratory-measured pairwise signals called the head-related impulse response (HRIR) which encode the sound source position in relation to the listener. In the laboratory an HRIR is captured by two in-the-ear microphones in a dummy head or human test subject: an impulsive sound is played from a loud speaker at some distance, horizontal, and vertical angle. Thus a personalized set of HRIRs with radius, azimuth angle, and elevation angle properties are captured for each test subject **Fig 1**. So to digitally construct a sound source that occurs 1 meter away at 0° azimuth and 0° elevation (in front, at eye level), the pure sound is convolved with the HRIR at $[0,0,1]$.

The goal of this project is to predict the associated azimuth angle of an unknown test subjects' HRIR using machine learning. The motivation for the project is to deepen my understanding of spatialized sound, auditory modelling, and machine learning applications for audio. This project is similar to a study done at Aalto's Acoustics Labs on predicting azimuth angle of a sound [1], and thus will borrow from the study's data preprocessing methods and feature selection.

In section 2 Problem Formulation, the raw HRIR data from the ARI database is explained. Section 3 Method lays out the preprocessing of raw data from the data set, and selection and design of two machine learning models, linear and polynomial regression, and multi-layer perceptron. Section 4 Results discusses the performance of the machine learning models.



(a) In-the-ear microphone



(b) Top-down measurement setup

Figure 1: (a) In-the-ear canal microphones. Photo courtesy of ARI HRTF database [2]. (b) Top-down view of sound source position in relation to test subject, nose indicates test subject facing top-of-page. Photo courtesy of [3].

2 Problem Formulation

2.1 Explanation of data set

The data is from the ARI database [4] containing the HRIRs from 177 subjects **Fig 2**. For the sake of simplicity, we are only interested in predicting the azimuth angle of an HRIR in the horizontal plane. For each subject 90 HRIRs were measured in the horizontal plane, so the number of data points is approximately $90 * 177 = 15930$. The data points are $\text{HRIR}[y, 0, 1.20]$, and the labels are the azimuth angles $y \in [-180, 180)$. Each HRIR is a pair of 5.33 ms long pressure signals, consisting of 256 samples **Fig 3**. Because each HRIR data point is large and ordered (2×256 sequential values per HRIR), the data in its raw form would complexify a machine learning task. The next section discusses how the raw data was preprocessed into a high-level representation suitable for a machine learning task.

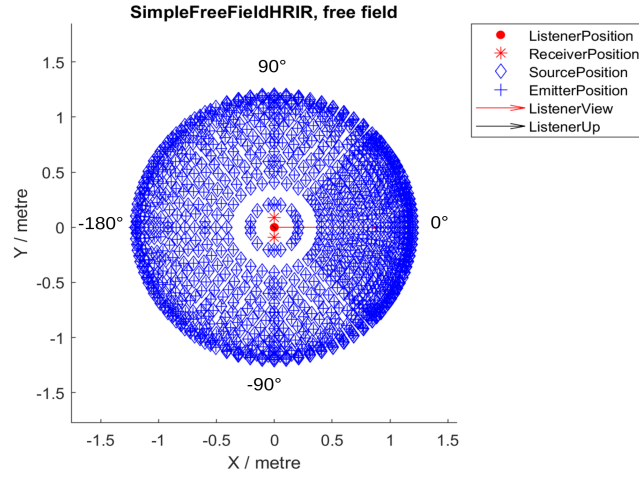


Figure 2: Top down view of HRIRs of an ARI test subject. Each blue diamond marks a source location $[x, y, z]$ for which an HRIR was measured. The HRIRs of interest are at the edge of the distribution.

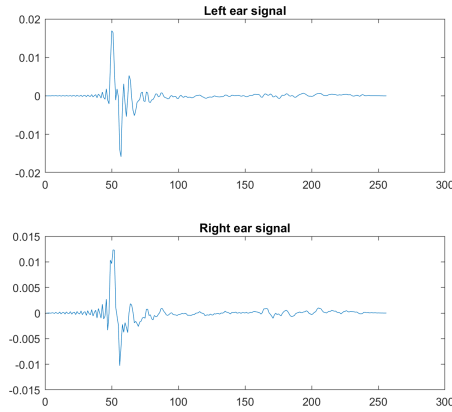


Figure 3: Example HRIR data point. Consists of a left and right pressure signal of length 256.

3 Method

3.1 Feature engineering and data preprocessing

Psychoacoustic studies have shown that differences between the signals arriving to the ears, interaural time difference ITD and interaural level difference ILD, are key measures correlated with sound localization. Because ITDs and ILDs are so important, ITDs and ILDs were chosen to be the features to represent the data points for the machine learning task. Additionally, representing a binaural signal with ITDs and ILDs removes irrelevant and distracting features in the raw HRIRs that the machine learning model may wrongly learn, and would result in small training error but large validation error.

The ITDs and ILDs were extracted from the HRIRs in Matlab using the methodology in the Acoustics Lab study [1]. To summarize, the HRIRs were convolved with 250ms pink noise to create binaural signals. Then, the binaural signals were passed through an auditory model from the Auditory Modeling Toolbox [5] to extract the ITDs and ILDs. Each HRIR was then transformed into a feature vector with 36 features: an ITD for each of the 18 cochlear frequency bands in the 50 Hz to 1.5 kHz range, and an ILD for each of the 18 cochlear frequency bands in the 1 kHz to 8 kHz range. See **Tab. 1** for five samples from the feature matrix X and label vector y .

	itd1	itd2	...	itd18	ild1	...	ild18		y
0	0.000038	0.000040	...	-0.000003	0.300998	...	3.535793		0.0
1	-0.000005	-0.000028	...	-0.000021	-0.420428	...	2.062511		2.5
2	-0.000043	-0.000071	...	-0.000040	-0.824443	...	1.798940		5.0
3	-0.000055	-0.000110	...	-0.000057	-1.081853	...	1.653616		7.5
4	-0.000148	-0.000181	...	-0.000079	-1.376703	...	0.315899		10.0
...									
[15928 rows x 37 columns]									

Table 1: First five rows of [15928 x 36] feature matrix X and [15928 x 1] label vector y .

This data was exported from matlab and imported into a python jupyter notebook. The data was split randomly into 70% training set and 30% test set. This split was chosen based on the widely used 80-20 split, however because the data set was quite large, I decided to increase the test set size. The training set and test set were normalized by removing the training set pre mean and scaling to training set unit variance using `StandardScaler()` from `sklearn.preprocessing`. In early stages of the project, it was found that normalization greatly improved the performance of the models.

3.2 ML methods

The machine learning task is a supervised-learning regression task, where given a feature vector representing an HRIR, predict the label representing the azimuth angle. This means the output of our machine learning models should be one value with a continuous range. In this project, we evaluate the performance of the linear regression and polynomial regression method against the multi-layer perceptron method. Mean square error was used to compare the validation error between models: it is useful because it weighs large deviations from the true labels more heavily.

Linear regression is used because the data shows some linear relationship; however, the data also shows some more complex behavior **Fig 4**, so polynomial regression method is attempted. Here, we try polynomial regression of degree one (linear regression), two, and three. The loss function during training was squared error, which is the most common lost function for simple regression models and is implemented in the scikit-learn's `sklearn.linear_model.LinearRegression` class. Then to evaluate generally how well logistic and polynomial regression models performed, k-fold cross-validation procedure with $k = 5$ folds was performed on each model.

There is a possibility that the underlying concept of the relationship between HRIR and azimuth angle cannot be polynomially represented, so a set of multi-layer perceptron (MLP), a fully connected class of feedforward artificial neural network (ANN), was trained as well. In this project, the hyperparameter number of hidden layers (1, 2, 3, or 4) was optimized. The number of neurons per layer was chosen by following the ANN design used in [1] which had 16 neurons per hidden layer. The loss function was square error which is a common loss function for MLPs, and it implemented with a stochastic gradient-based optimizer 'adam' in `sklearn.neural_network.MPLRegressor`.

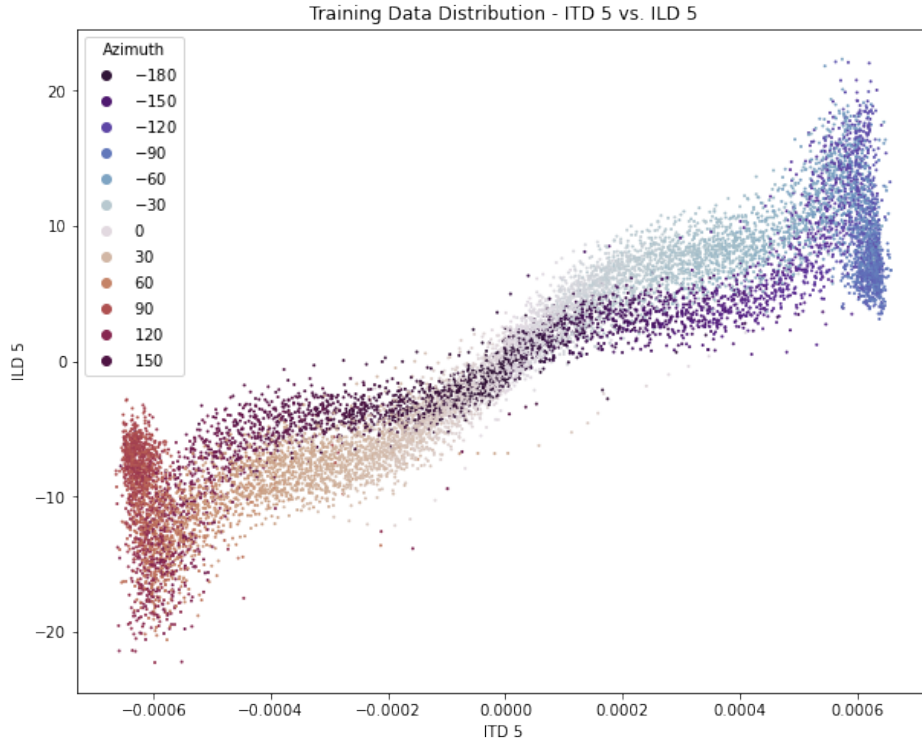


Figure 4: Scatterplot of training data with color corresponding to label. Features 'ITD 5' and 'ILD 5' on the axes.

4 Results and Conclusion

4.1 Linear and Polynomial Regression

A linear regression model, 2nd degree, and 3rd degree polynomial regression models were implemented. The training mean squared errors and validation mean squared errors of K-fold cross validation with $k = 5$ were averaged to give an average training error and validation error for each model, see **Tab 2**. As stated in Section 3.1, 70% of the data was training data which went into k-fold cross validation. The remaining 30% was set aside to be the test set. The trained models then predicted the test set, and generated a test error. We order the performance of each model based on the average validation mean squared error. Unfortunately k-fold cross validation could not be performed on the 3rd degree polynomial regression model because of excessive memory usage, so the values reported in the table are the training error and test error of one model.

Then, the models were tested on the test set to get a test error. The linear regression model performed well with similar average training and validation error in the 1740-1760 range. We see that the 2nd degree polynomial regression model performed the best because it has the lowest average validation error of 1455.7 and test error of 1276.8. And lastly, the 3rd degree polynomial had the lowest training error of 374.5, but the test error was 7 fold over the other models' test errors meaning that the 3rd degree polynomial regression model was overfitting.

Error of nth degree Polynomial Regression			
Degree	Avg. Training Error	Avg. Validation Error	Test Error
1	1745.8	1756.7	1658.0
2	1276.2	1455.7	1276.8
3	374.5	—	76288.6

Table 2: Linear and polynomial regression table of average training, average validation, and test error.

4.2 Multi-layer Perceptron Regression

Four multi-layer perceptron regression models were implemented. Similar to the polynomial regression analysis, the average training mean squared errors and validation mean squared errors were computed in K-fold cross validation with $k = 5$ folds (**Tab 3**). The MLP predicted on the same test set as above. We see that the average validation error of the 2-layer MLP was the lowest at 776.9, and that the validation error began to increase from 3-layer MLP onward, which hints at overfitting. However because the average validation error values and test error values conflict yet are very close in magnitude, visualization of the predicted labels would give more insight on the best performing MLP model. This is discussed in the next subsection.

Error of Multi-layer Perceptron Regression			
Num. Hidden Layers	Avg. Training Error	Avg. Validation Error	Test Error
1	1429.3	1456.7	1329.6
2	571.0	776.8	673.9
3	339.0	803.4	568.9
4	393.7	904.4	644.1

Table 3: Linear and polynomial regression table of testing and validation error.

4.3 Discussion & Conclusion

The MLP regression on average performed better than linear and polynomial regression. Only the 2nd degree polynomial regression's validation and test error matched those of the worst performing MLP (1-layer). The data above suggests the best model according to lowest average validation error was the 2-layer MLP. However, the visualization suggests instead the 3-layer MLP outperformed the 2-layer MLP on the test set.

When plotting the true label vs. the predicted label of each of the polynomial regression models, we can see that the 2nd degree polynomial regression model performed best. The plot of the 3rd degree polynomial regression shows predicted values that are completely out of the label range of $[-180, 180]$, a clear sign of overfitting. Linear and polynomial regression have difficulty predicting the extreme labels which can be seen in the large deviation from the gray line at the extremes **Fig 5.a & 5.b**. This makes sense looking at the training set distribution **Fig 4**, the point spread suggests that features with label around 0 are very similar to features with label $= +/-180$.

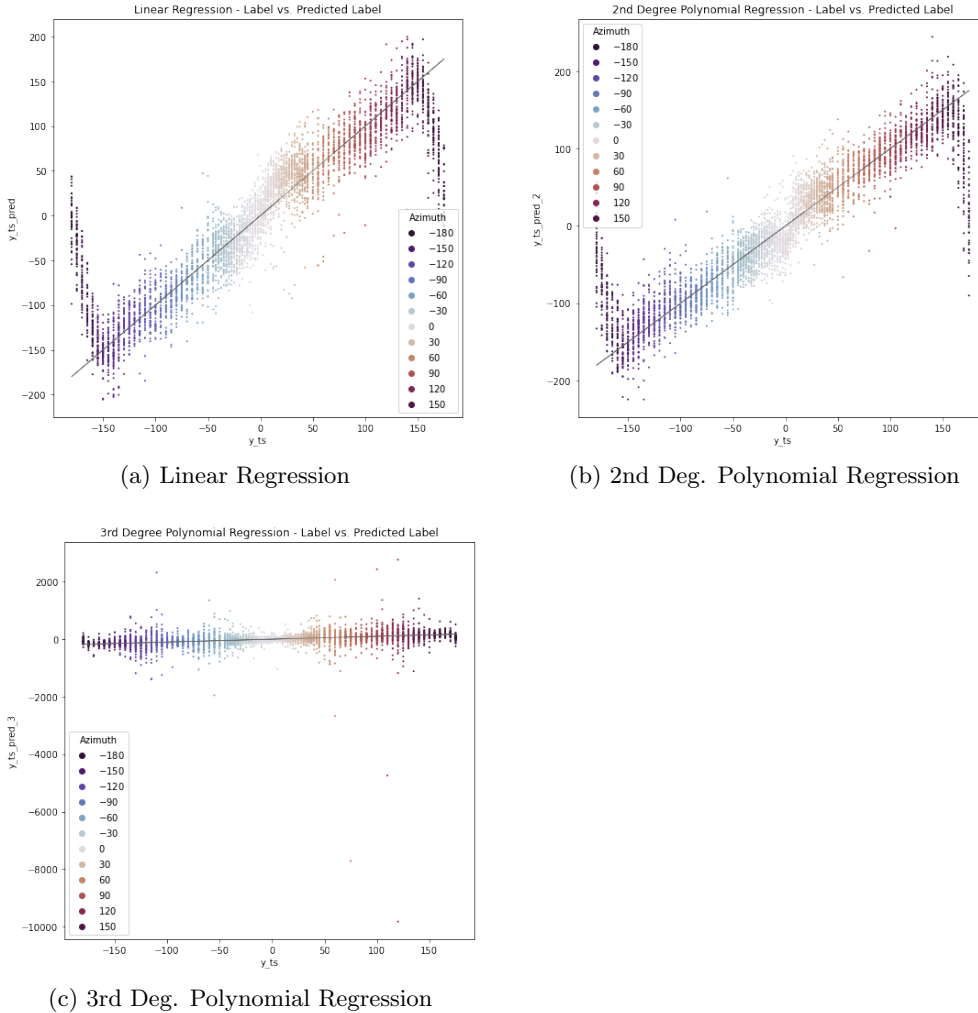


Figure 5: The graphs show the test labels on the x-axis and the predicted test labels on the y-axis for the (a) linear regression model (b) 2nd degree polynomial model (c) 3rd degree polynomial model. The gray line $x = y$ line shows where the data points would land if the model perfectly predicted the labels.

In comparison, the MLP regression plots in **Fig 6** show much better accuracy when the MLP has 3 hidden layers or 4 hidden layers—the point spread is much denser, centered closely on the optimal diagonal line. The 3-layer MLP show better performance than the 2 layer MLP. The plot of the 4-layer MLP shows overfitting, where some predicted value is ~ 300 , which is out of label's range. Similar to linear and polynomial regression, all the MLP models have difficulty predicting the labels at ± 180 .

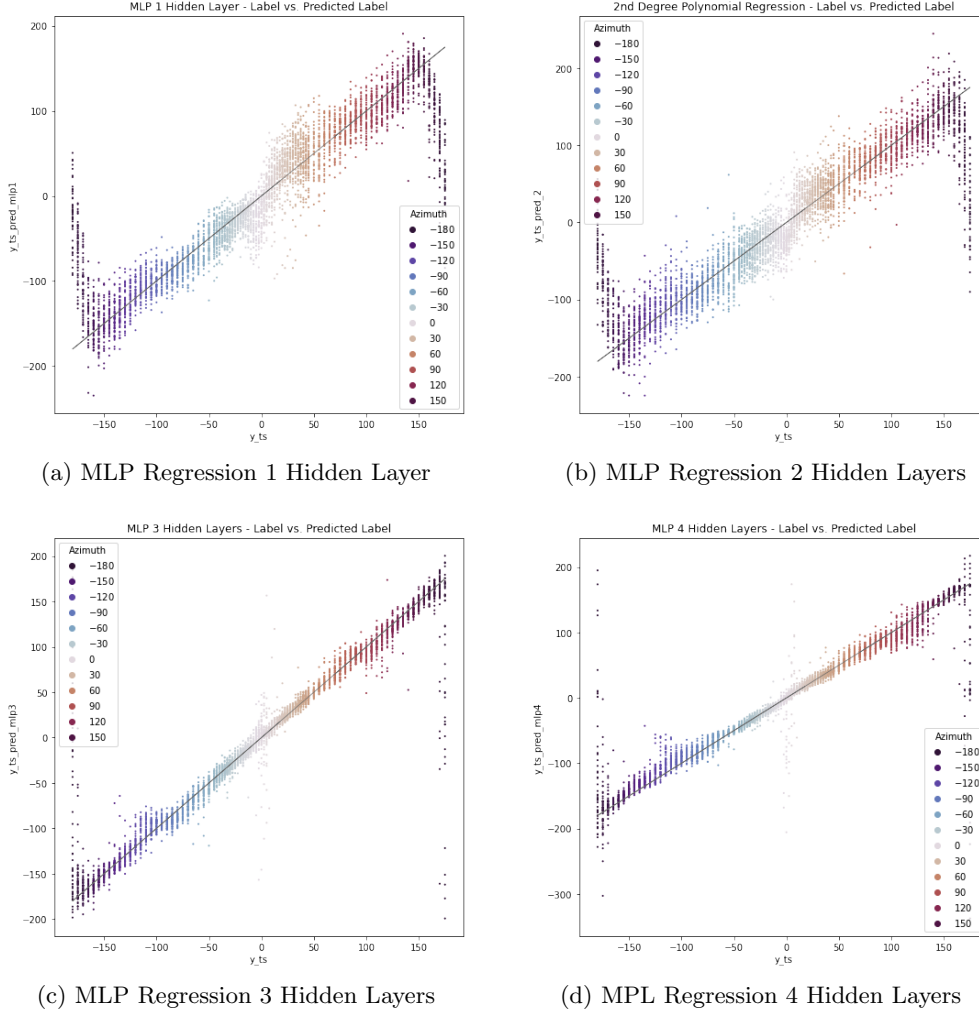


Figure 6: The graphs show the actual labels on the x-axis and the predicted labels on the y-axis for the MLP regression models with (a) 1 (b) 2 (c) 3, and (d) 4 hidden layers. The gray line $x = y$ line shows where the data points would land if the model perfectly predicted the labels.

Overall this report found that 1.) MLP regression outperformed linear and polynomial regression and 2.) A visualization of the data helped us pick the best model, the 3-layer MLP model, over a seemingly better performing 2-layer MLP model. A further exploration of the hyperparameters for the MLP model would be advantageous to fine-tune this model. This includes optimizing the number of neurons, activation function (here we used ReLU, but there is logistic sigmoid, tanh), solver, etc. A deep learning system might achieve better results than a MLP. AN interesting future direction would be to use these models trained on the ARI HRTF Database to predict the labels of HRIRs from other databases.

References

- [1] Lladó, Pedro, Hyvärinen, Petteri, and Pulkki, Ville, “Auditory model-based estimation of the effect of head-worn devices on frontal horizontal localisation,” *Acta Acust.*, vol. 6, p. 1, 2022. [Online]. Available: <https://doi.org/10.1051/aacus/2021056>
- [2] Acoustics Research Institute (ARI), “In-the-ear microphone for hrtf,” 2022. [Online]. Available: https://www.oeaw.ac.at/fileadmin/Institute/ISF/IMG/software/hrtf_main_ear1.jpg
- [3] SOFA Conventions, user:Petibub, “HRTF azimuth,” 2013. [Online]. Available: <http://www.sofaconventions.org/mediawiki/index.php/SimpleFreeFieldHRIR>
- [4] Acoustics Research Institute (ARI), “Ari hrtf database,” 2014. [Online]. Available: <https://www.oeaw.ac.at/en/isf/das-institut/software/hrtf-database>
- [5] P. Majdak, C. Hollomey, and the AMT team, “Auditory Modeling Toolbox,” 2003-2022. [Online]. Available: <http://amttoolbox.org/>

Matlab Code

Data preprocessing code

```
1 addpath('HUTUBS/');
2 addpath('functions/');
3 clear all;
4 close all;
5
6 %% Example HRIR from ARI HRTF Database:
7 hrtf2 = SOFALoad('hrtf b_nh.sofa');
8 fs = 48000;
9 soundInput = pinknoise(0.25*fs);
10
11 %% demo script
12 % Start SOFA
13 SOFASstart;
14 % Display some information about the impulse response
15 % Plot a figure with the measurement setup
16 SOFASplotGeometry(hrtf2);
17
18 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
19 % Data processing - extract ITDs and ILDs for every horizontal HRIR in ARI
20 % HRTF Database
21 X = [];
22 y = [];
23 count = 0;
24 db = [];
25
26 for i = 1:830
27     sofaFileName = append('hrtf b_nh', num2str(i), '.sofa');
28     hrtf = extractHorizontalIirs(sofaFileName, fs);
29     % if sofaFileName is found in ARI database, compute binaural features
30     if hrtf.tf ~= 0
31         % Find ITDs and ILDs
32         binFeats = transpose(llado2022_binauralfeats(hrtf.IR, soundInput, hrtf
33 .fs));
34         disp(['binFeats found for person_ID ', num2str(i)])
35         count = count + 1;
36         p_ID = ones(length(hrtf.azi), 1)*i;
37         p_count = ones(length(hrtf.azi), 1)*count;
38
39         % Add to database db
40         dataCombined = cat(2, p_count, p_ID, binFeats, hrtf.azi);
41         db = cat(1, db, dataCombined);
42
43         % Write data so far
44         if i==30 || i==60 || i==120 || i==170 || i==259 || i==830
45             filename = '.xlsx';
46             dataCombined = cat(binFeats, y);
47             writematrix(db)
48             disp(['File wrote. last person id # ', num2str(i)])
49         end
50     end
51 end
```

extractHorizontalirs function

```
1 function [data] = extractHorizontalirs(sofaFileName,fs)
2 %   Input parameters:
3 %       sofaFileName    : hrir set in sofa format
4 %       fs              : sample frequency
5 %
6 %   Output parameters:
7 %       data            : Impulse response according to the following matrix
8 %                       dimensions: direction x time x channel/ear
9 hrirname = char(sofaFileName);
10
11 % try to find sofaFileName in ari database
12 try
13     hrtf = SOFALoad(['http://sofacooustics.org/data/database/ari/',hrirname])
14     ;
15     if (fs ~= hrtf.Data.SamplingRate)
16         amt_disp('Sample Frequency mismatch');
17     end
18     % Calculate the source position from a listener point of view
19     apparentSourceVector = SOFACalculateAPV(hrtf);
20     % Create an array of indices for IRs that have elevation=0
21     idx = find(hrtf.SourcePosition(:,2)==0);
22
23     % Create an empty array to store labels
24     data.azi = zeros(length(idx),1);
25
26     for i = 1:length(idx)
27         data.IR(i,:,:)=squeeze(hrtf.Data.IR(idx(i),:,:));
28         data.azi(i,1) = apparentSourceVector(idx(i),1);
29     end
30     data.IR = permute(data.IR,[1,3,2]);
31     data.fs = hrtf.Data.SamplingRate;
32     data.tf = 1;
33 catch
34     data.tf = 0;
35 end
```

project-Copy1

October 9, 2022

```
[1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.tree import export_text, plot_tree, DecisionTreeClassifier,
    export_graphviz
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, accuracy_score
from sklearn.neural_network import MLPRegressor
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import KFold, cross_val_score, cross_validate
```

Fontconfig error: Cannot load default config file: No such file: (null)

```
[2]: db = pd.read_csv("db.txt",
    sep=',', header=None, names=['p_count', 'p_ID', 'itd1', 'itd2', 'itd3', 'itd4', 'itd5', 'itd6', 'itd7',
    'itd8', 'itd9', 'itd10', 'itd11', 'itd12', 'itd13', 'itd14', 'itd15', 'itd16',
    'itd17', 'itd18',
    'ild1', 'ild2', 'ild3', 'ild4', 'ild5', 'ild6', 'ild7', 'ild8',
    'ild9', 'ild10', 'ild11', 'ild12', 'ild13', 'ild14', 'ild15', 'ild16',
    'ild17', 'ild18', 'y'])
db.head()
```

```
[2]:
```

	p_count	p_ID	itd1	itd2	itd3	itd4	itd5 \
0	1	2	0.000038	0.000040	0.000007	0.000007	-8.680556e-07
1	1	2	-0.000005	-0.000028	-0.000016	-0.000011	-2.951389e-05
2	1	2	-0.000043	-0.000071	-0.000036	-0.000030	-5.555556e-05
3	1	2	-0.000055	-0.000110	-0.000052	-0.000042	-7.378472e-05
4	1	2	-0.000148	-0.000181	-0.000090	-0.000071	-1.024306e-04

	itd6	itd7	itd8	...	ild10	ild11	ild12	ild13	\
0	-0.000003	-0.000003	0.000000	...	0.300998	1.338371	-0.761319	-0.701190	
1	-0.000021	-0.000031	-0.000034	...	-0.420428	-0.382010	-2.288078	-2.029885	
2	-0.000040	-0.000060	-0.000064	...	-0.824443	-1.468890	-2.977154	-2.810987	
3	-0.000057	-0.000088	-0.000102	...	-1.081853	-2.366840	-3.670521	-3.631215	
4	-0.000079	-0.000121	-0.000139	...	-1.376703	-3.228623	-4.668062	-4.832840	

	ild14	ild15	ild16	ild17	ild18	y
0	-1.656613	-2.279561	-1.311583	5.662943	3.535793	0.0
1	-2.675837	-4.257018	-3.957735	3.715287	2.062511	2.5
2	-3.319359	-5.452643	-5.916478	2.338014	1.798940	5.0
3	-3.960667	-6.686579	-7.835403	0.534389	1.653616	7.5
4	-4.933389	-7.897229	-9.212499	-0.512847	0.315899	10.0

[5 rows x 39 columns]

```
[6]: X = db[db.columns]
X.drop(['p_count', 'p_ID', 'y'], axis=1, inplace=True)
X = X.to_numpy().reshape(-1,36)

y = db['y'].to_numpy()
```

```
[7]: X_tr, X_ts, y_train, y_test = train_test_split(X, y, test_size=0.3,
↳ random_state=42)
```

```
[8]: # Standardize data using sklearn.preprocessing.StandardScaler
scaler = StandardScaler()
scaler.fit(X_tr)
X_train = scaler.transform(X_tr)
X_test = scaler.transform(X_ts)
print(np.shape(X_train))

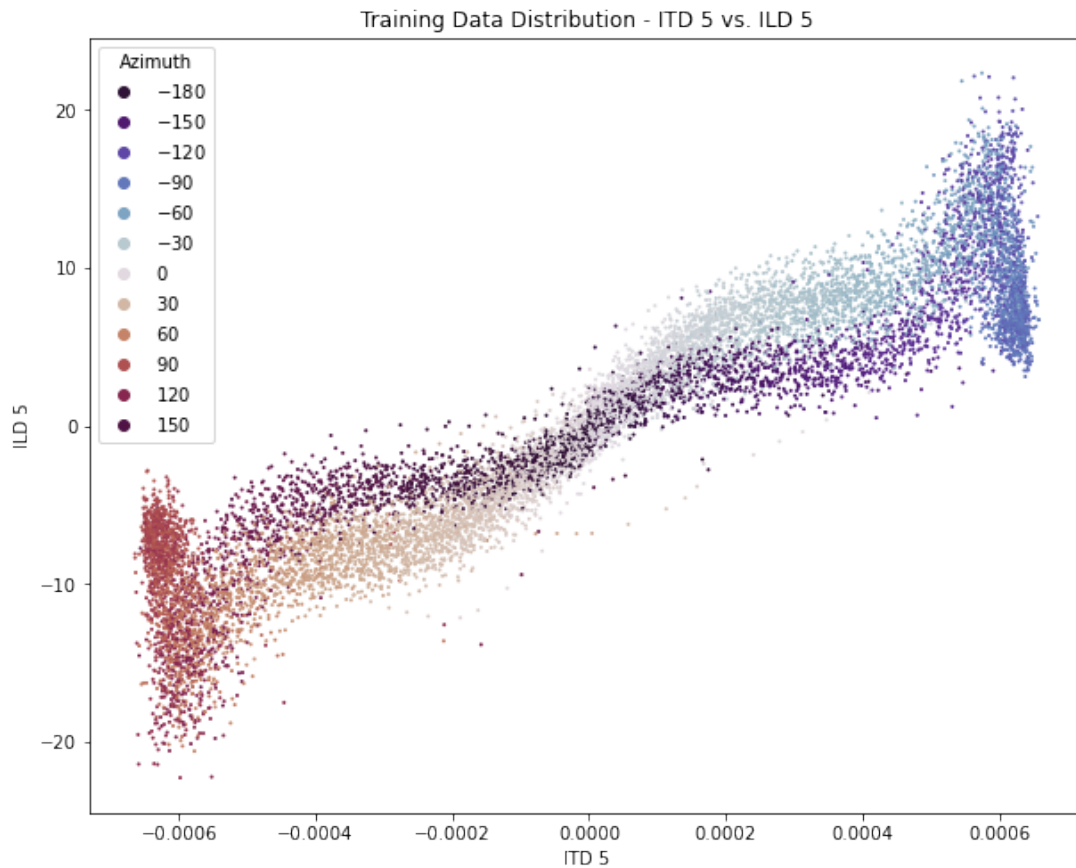
linear_tr_error = []
linear_ts_error = []
```

(11149, 36)

0.1 Dataset Visualization

```
[13]: plt.figure(figsize=(10, 8))
# convert the labels to numbers, each will be assigned a separate color based
↳ on the cmap specified
colors = [int(x) for x in y_train]
sc = plt.scatter(X_tr[:, 5], X_tr[:, 23], s=1, c=colors,
↳ cmap='twilight_shifted')
plt.xlabel("ITD 5")
plt.ylabel("ILD 5")
```

```
plt.legend(*sc.legend_elements(), title='Azimuth')
plt.title("Training Data Distribution - ITD 5 vs. ILD 5")
plt.show()
```



0.2 Linear and polynomial regression

```
[14]: # Linear regression
# K-fold cross validation: k = 5
standard_scaler = StandardScaler()
lin_regr = LinearRegression(fit_intercept=False)

pipeline = make_pipeline(standard_scaler, lin_regr)
# perform k-Fold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=2)
# k-fold cross-validation conduction
cv_results = cross_validate(pipeline, # Pipeline
                           X_tr, # Feature matrix
                           y_train, # Target vector
```

```

cv=kf, # Cross-validation technique
scoring="neg_mean_squared_error", # Loss function
n_jobs=-1,
return_train_score=True) # Use all CPU cores

```

```
[18]: np.mean(cv_results['test_score'])
```

```
[18]: -1756.7045943597955
```

```
[17]: np.mean(cv_results['train_score'])
```

```
[17]: -1745.7972008378358
```

```

[7]: # Simple Linear Regression on scaled features
lin_regr = LinearRegression(fit_intercept=False)
lin_regr.fit(X_train,y_train)

y_tr_pred = lin_regr.predict(X_train)
tr_error = mean_squared_error(y_train,y_tr_pred)

y_ts_pred = lin_regr.predict(X_test)
ts_error = mean_squared_error(y_test,y_ts_pred)

print('The training error is: ', tr_error)    # print the training error
print('The validation error is: ', ts_error)  # print the testing error
print("w1 = ", lin_regr.coef_)               # print the learnt w1
print("w0 = ", lin_regr.intercept_)          # print the learnt w0

linear_tr_error.append(tr_error)
linear_ts_error.append(ts_error)

```

The training error is: 1746.8719156461407

The validation error is: 1658.010794175003

w1 = [-1.79945895e+01 -6.07822359e+01 -7.15427562e+00 -2.95237550e+01

1.73847874e+02 1.43796809e+02 -7.10237258e+02 4.54075087e+02

-7.26507718e+01 -4.02140416e+01 3.47538661e+01 1.73942253e+00

8.53341295e+00 -2.58217759e+00 -1.10754677e+01 -1.19923541e+01

-4.62903838e+00 3.55354421e+00 -1.06209296e+01 -2.55886105e+01

-7.31369286e+00 4.05944305e+01 -1.03082703e+01 1.08047925e+01

-2.77004147e+01 3.38440272e+01 2.08933598e+01 -2.66212120e+01

2.11885768e+01 2.98180430e+00 5.96227633e+00 1.36744539e+01

2.34255921e+01 1.16834025e+01 -1.00173597e+01 -3.71898197e-01]

w0 = 0.0

```

[ ]: # 2nd Degree Polynomial regression
# K-fold cross validation: k = 5
standard_scaler = StandardScaler()

```

```

lin_regr = LinearRegression(fit_intercept=False)
poly = PolynomialFeatures(degree=3)
X_train_poly = poly.fit_transform(X_tr)

pipeline = make_pipeline(standard_scaler, lin_regr)
# perform k-Fold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=2)
# k-fold cross-validation conduction
cv_results = cross_validate(pipeline, # Pipeline
                             X_train_poly, # Feature matrix
                             y_train, # Target vector
                             cv=kf, # Cross-validation technique
                             scoring="neg_mean_squared_error", # Loss function
                             n_jobs=-1,
                             return_train_score=True) # Use all CPU cores

```

```
[24]: np.mean(cv_results['test_score'])
```

```
[24]: -1455.708752958774
```

```
[25]: np.mean(cv_results['train_score'])
```

```
[25]: -1276.1800424723567
```

```

[8]: degrees = [2]

for i in degrees:
    # Transform feature vector into polynomial features
    poly = PolynomialFeatures(degree=i)
    X_train_poly = poly.fit_transform(X_train)
    X_test_poly = poly.transform(X_test)

    # Fit a linearRegression model. Validate on polynomial train and polynomial_
    ↪ test
    lin_regr = LinearRegression()
    lin_regr.fit(X_train_poly, y_train)

    y_tr_pred_2 = lin_regr.predict(X_train_poly)
    tr_error_2 = mean_squared_error(y_train, y_tr_pred_2)

    y_ts_pred_2 = lin_regr.predict(X_test_poly)
    ts_error_2 = mean_squared_error(y_test, y_ts_pred_2)

    # Append errors to arrays
    linear_tr_error.append(tr_error_2)
    linear_ts_error.append(ts_error_2)

```

```
[9]: degrees = [3]

for i in degrees:
    # Transform feature vector into polynomial features
    poly = PolynomialFeatures(degree=i)
    X_train_poly = poly.fit_transform(X_train)
    X_test_poly = poly.transform(X_test)

    # Fit a linearRegression model. Validate on polynomial train and polynomial
    ↪ test
    lin_regr = LinearRegression()
    lin_regr.fit(X_train_poly, y_train)

    y_tr_pred_3 = lin_regr.predict(X_train_poly)
    tr_error = mean_squared_error(y_train, y_tr_pred_3)

    y_ts_pred_3 = lin_regr.predict(X_test_poly)
    ts_error = mean_squared_error(y_test, y_ts_pred_3)

    # Append errors to arrays
    linear_tr_error.append(tr_error)
    linear_ts_error.append(ts_error)
```

```
[15]: errors = {"poly degree": [1, 2, 3],
                "linear_train_errors": linear_tr_error,
                "linear_test_errors": linear_ts_error,
                }
pd.DataFrame({ key: pd.Series(value) for key, value in errors.items()})
```

/tmp/ipykernel_122/3709052835.py:5: FutureWarning: The default dtype for empty Series will be 'object' instead of 'float64' in a future version. Specify a dtype explicitly to silence this warning.

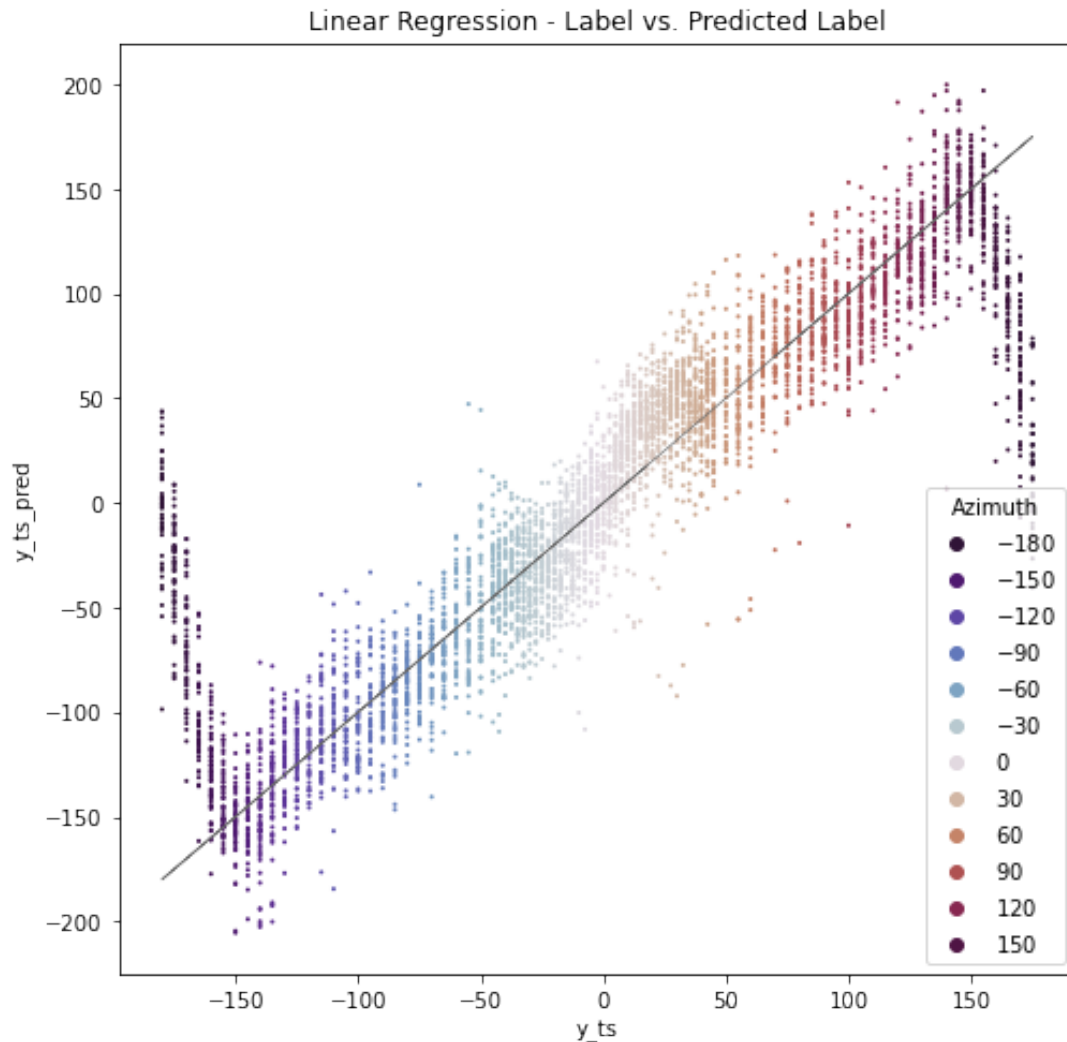
```
pd.DataFrame({ key: pd.Series(value) for key, value in errors.items()})
```

```
[15]:
```

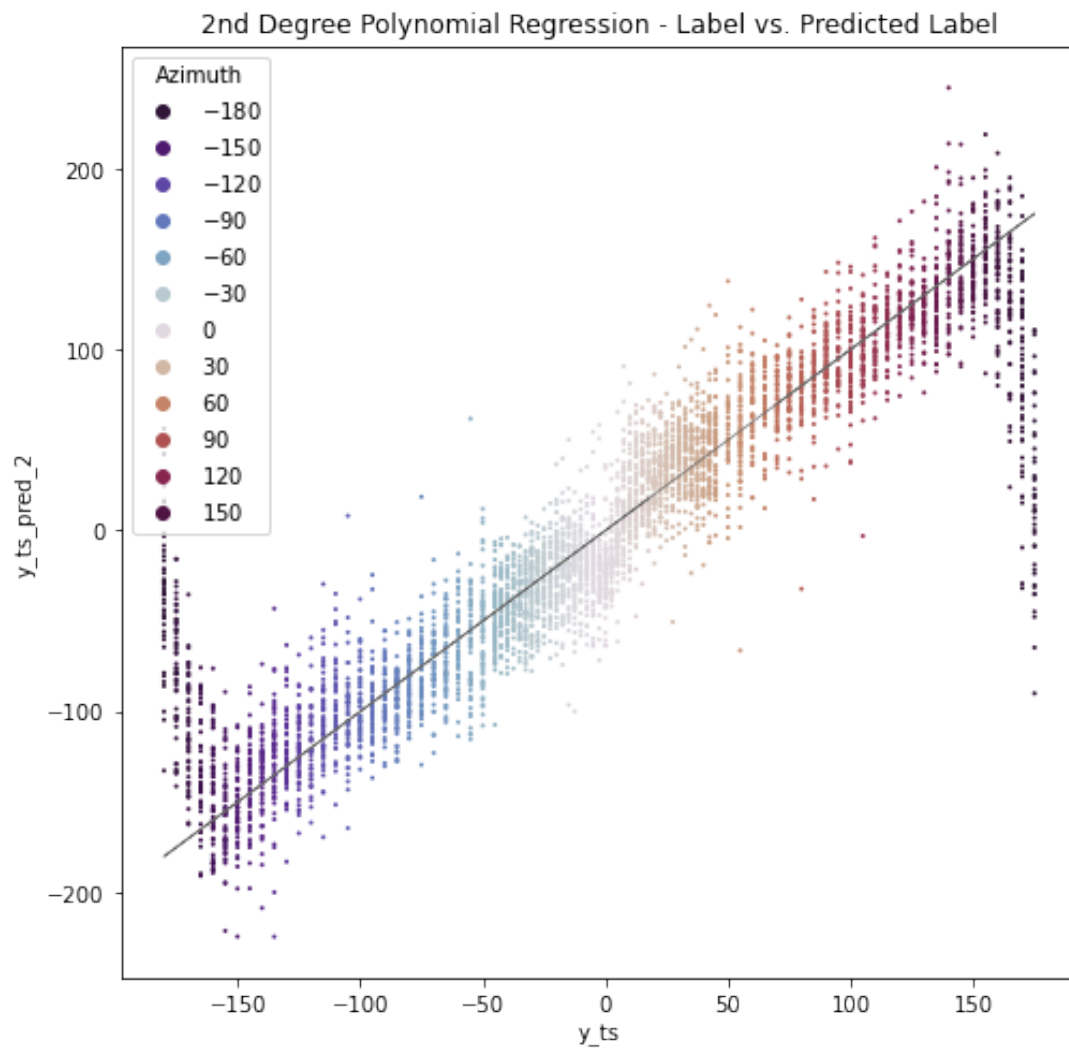
	poly degree	linear_train_errors	linear_test_errors
0	1	NaN	NaN
1	2	NaN	NaN
2	3	NaN	NaN

```
[10]: plt.figure(figsize=(8, 8))
# convert the labels to numbers, each will be assigned a separate color based
↪ on the cmap specified
colors = [int(x) for x in y_test]
sc = plt.scatter(y_test, y_ts_pred, s=1, c=colors, cmap='twilight_shifted')
plt.plot(y_test, y_test, color='dimgray', label='y_test=y_test', linewidth=0.7)
plt.xlabel("y_ts")
plt.ylabel("y_ts_pred")
```

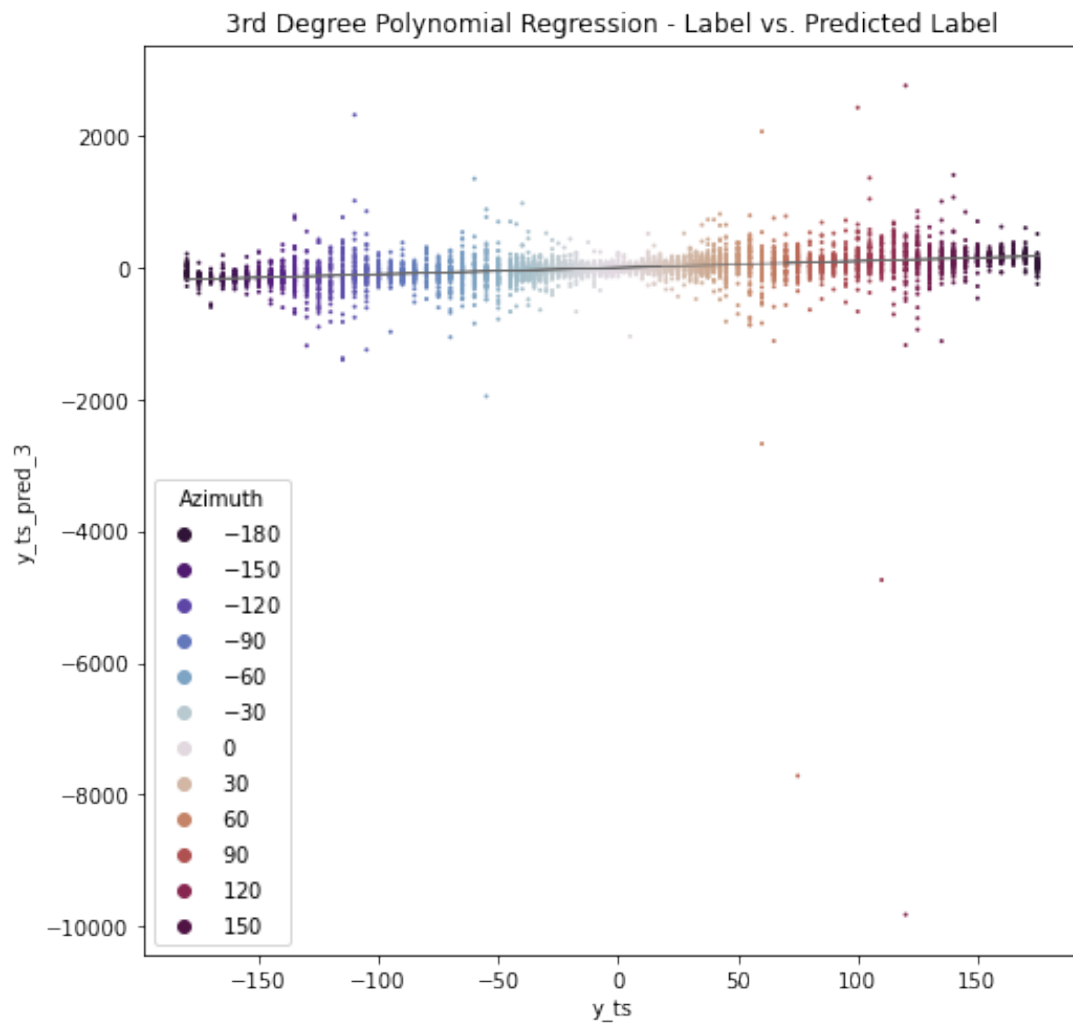
```
plt.legend(*sc.legend_elements(), title='Azimuth')
plt.title("Linear Regression - Label vs. Predicted Label")
plt.show()
```



```
[11]: plt.figure(figsize=(8, 8))
colors = [int(x) for x in y_test]
sc = plt.scatter(y_test, y_ts_pred_2, s=1, c=colors, cmap='twilight_shifted')
plt.plot(y_test, y_test, color='dimgray', label='y_test=y_test', linewidth=0.7)
plt.xlabel("y_ts")
plt.ylabel("y_ts_pred_2")
plt.legend(*sc.legend_elements(), title='Azimuth')
plt.title("2nd Degree Polynomial Regression - Label vs. Predicted Label")
plt.show()
```



```
[12]: plt.figure(figsize=(8, 8))
      colors = [int(x) for x in y_test]
      sc = plt.scatter(y_test, y_ts_pred_3, s=1, c=colors, cmap='twilight_shifted')
      plt.plot(y_test, y_test, color='dimgray', label='y_test=y_test', linewidth=0.7)
      plt.xlabel("y_ts")
      plt.ylabel("y_ts_pred_3")
      plt.legend(*sc.legend_elements(), title='Azimuth')
      plt.title("3rd Degree Polynomial Regression - Label vs. Predicted Label")
      plt.show()
```



0.3 Multi-layer Perceptron Regression

```
[9]: # ANN network parameters
# 1 Hidden layer
# 16 Neurons, fully connected
# Levenberg-marquardt backpropagation.

num_layers = [1,2,3,4]
num_neurons = 16

mlp_tr_error = []
mlp_ts_error = []
# np.array = Store y_pred values for each mlp option
mlp_tr_pred = np.zeros((len(y_train),len(num_layers)))
```

```

mlp_ts_pred = np.zeros((len(y_test),len(num_layers)))

for i, num in enumerate(num_layers):
    hidden_layer_sizes = tuple([num_neurons]*num)

    mlp_regr = MLPRegressor(hidden_layer_sizes=hidden_layer_sizes,
↪max_iter=2000, random_state=42)
    mlp_regr.fit(X_train, y_train)

    y_tr_pred_mlp = mlp_regr.predict(X_train)
    tr_error = mean_squared_error(y_train, y_tr_pred_mlp)

    y_ts_pred_mlp = mlp_regr.predict(X_test)
    ts_error = mean_squared_error(y_test, y_ts_pred_mlp)

    mlp_tr_error.append(tr_error)
    mlp_ts_error.append(ts_error)
    mlp_tr_pred[:, i] = y_tr_pred_mlp
    mlp_ts_pred[:, i] = y_ts_pred_mlp

```

```

[10]: MLPerrors = {"MLP layers":num_layers,
                  "MLP_train_errors":mlp_tr_error,
                  "MLP_val_errors":mlp_ts_error,
                  }
pd.DataFrame({ key:pd.Series(value) for key, value in MLPerrors.items()})

```

```

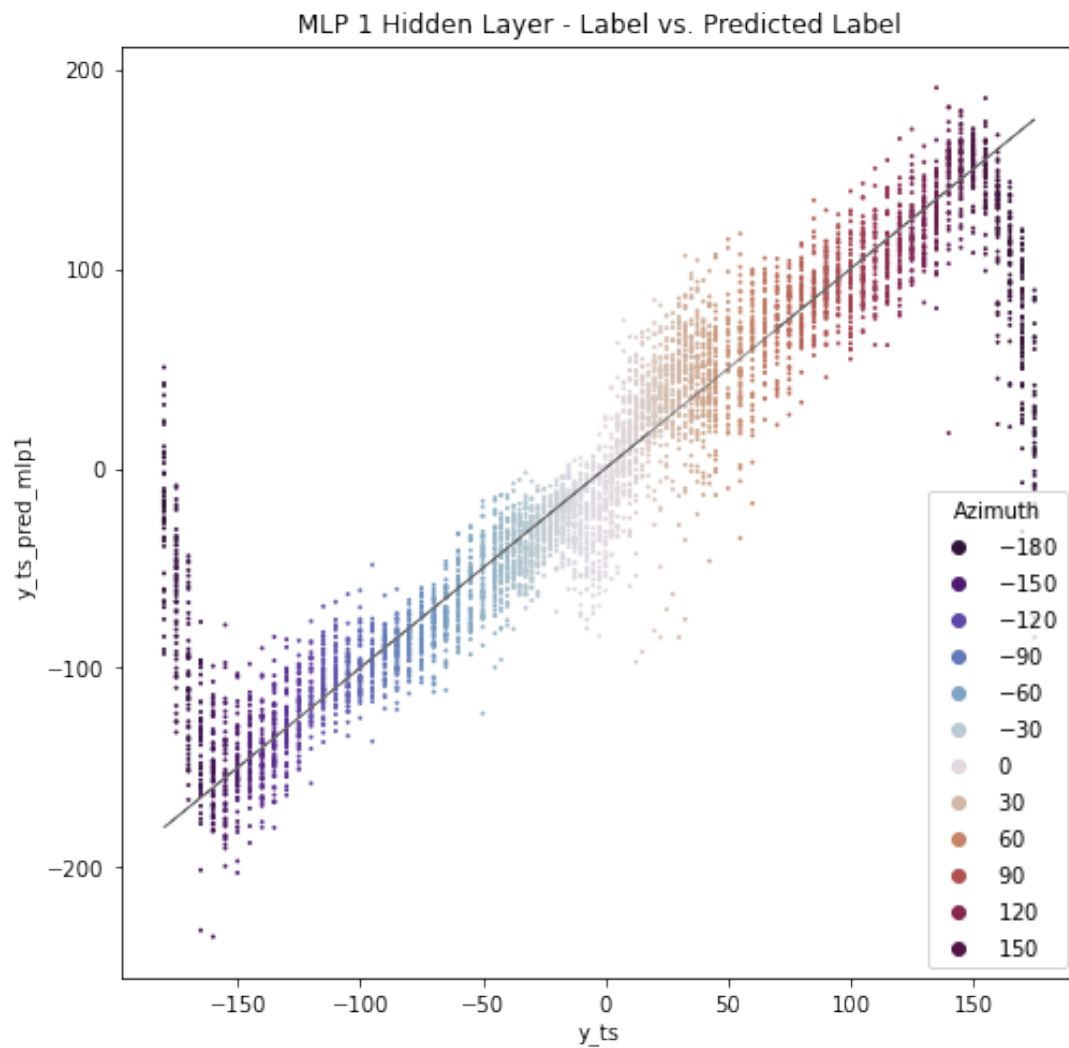
[10]:
  MLP layers  MLP_train_errors  MLP_val_errors
0          1       1398.432403       1329.630549
1          2        631.516809        673.911977
2          3        269.654495        568.935319
3          4        331.192819        644.140611

```

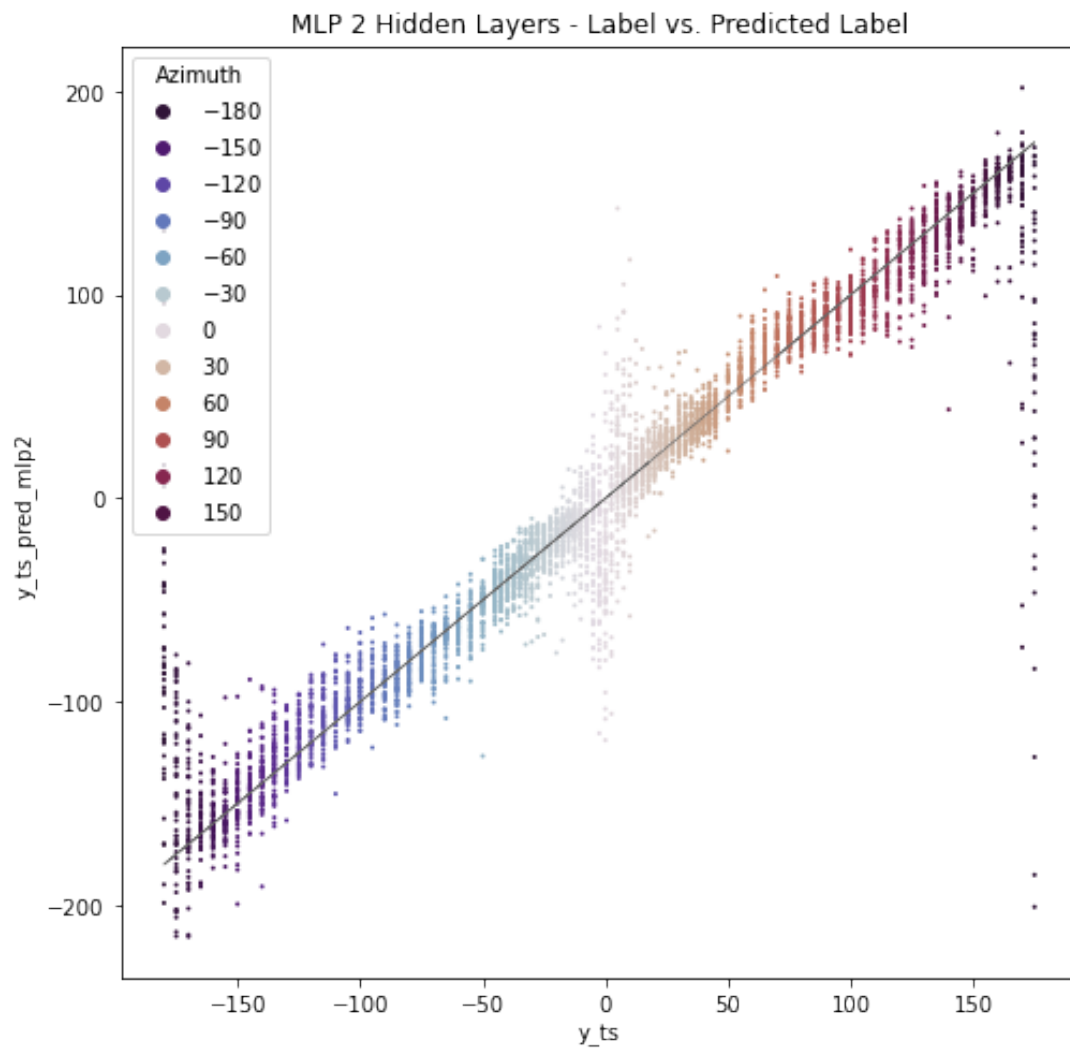
```

[11]: plt.figure(figsize=(8, 8))
      colors = [int(x) for x in y_test]
      sc = plt.scatter(y_test, mlp_ts_pred[:, 0], s=1, c=colors,
↪cmap='twilight_shifted')
      plt.plot(y_test, y_test, color='dimgray', label='y_test=y_test', linewidth=0.7)
      plt.xlabel("y_ts")
      plt.ylabel("y_ts_pred_mlp1")
      plt.legend(*sc.legend_elements(), title='Azimuth')
      plt.title("MLP 1 Hidden Layer - Label vs. Predicted Label")
      plt.show()

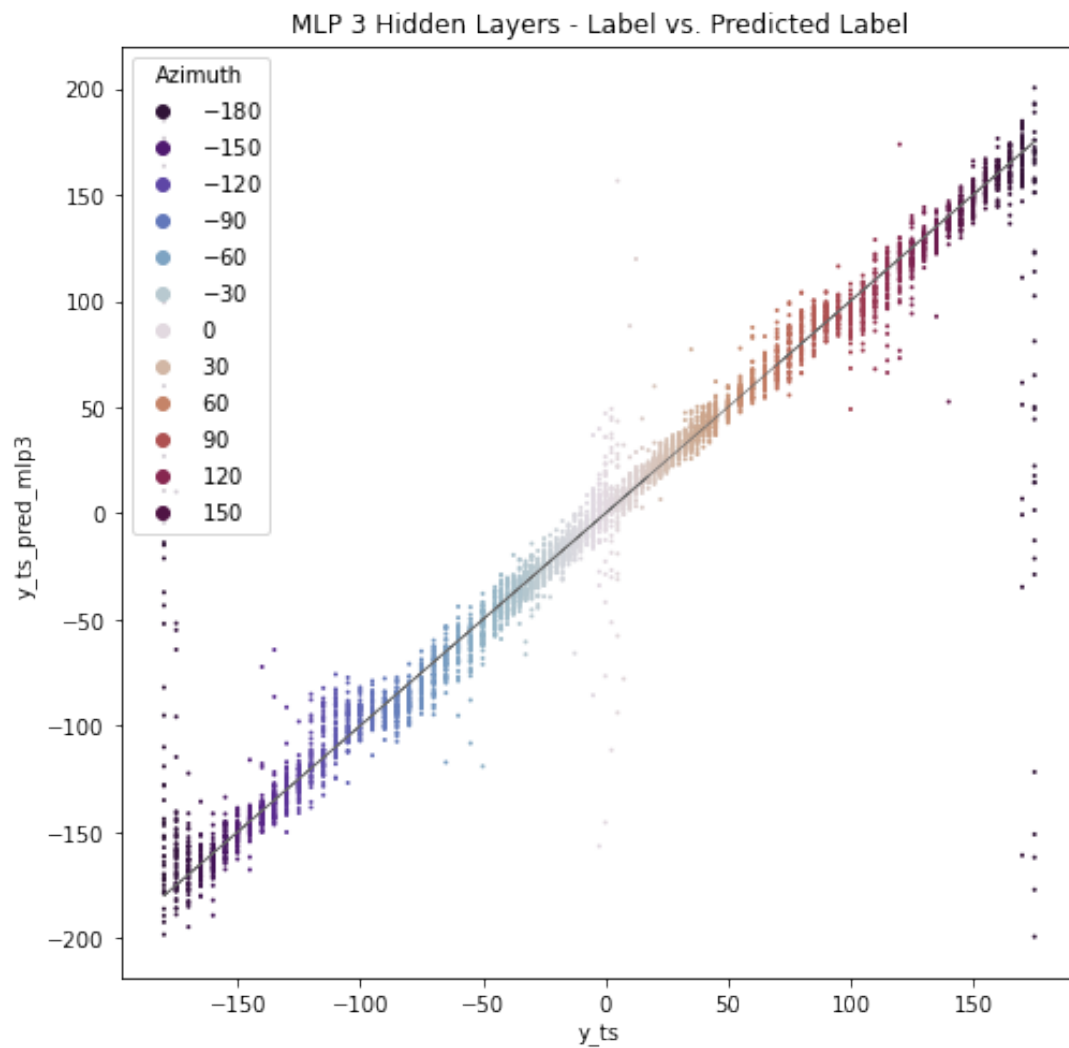
```



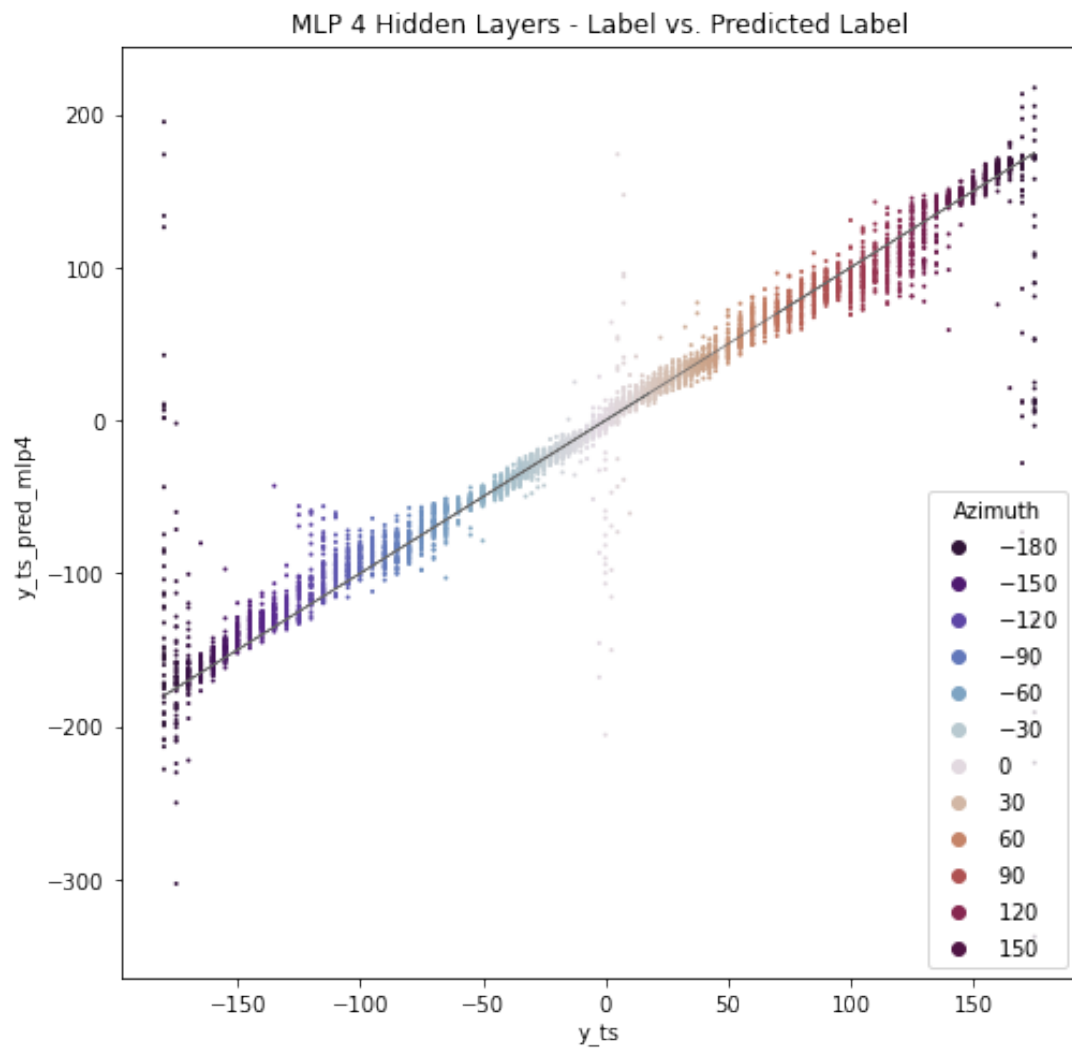
```
[12]: plt.figure(figsize=(8, 8))
      colors = [int(x) for x in y_test]
      sc = plt.scatter(y_test, mlp_ts_pred[:, 1], s=1, c=colors,
                      cmap='twilight_shifted')
      plt.plot(y_test, y_test, color='dimgray', label='y_test=y_test', linewidth=0.7)
      plt.xlabel("y_ts")
      plt.ylabel("y_ts_pred_mlp2")
      plt.legend(*sc.legend_elements(), title='Azimuth')
      plt.title("MLP 2 Hidden Layers - Label vs. Predicted Label")
      plt.show()
```



```
[13]: plt.figure(figsize=(8, 8))
      colors = [int(x) for x in y_test]
      sc = plt.scatter(y_test, mlp_ts_pred[:, 2], s=1, c=colors,
                        cmap='twilight_shifted')
      plt.plot(y_test, y_test, color='dimgray', label='y_test=y_test', linewidth=0.7)
      plt.xlabel("y_ts")
      plt.ylabel("y_ts_pred_mlp3")
      plt.legend(*sc.legend_elements(), title='Azimuth')
      plt.title("MLP 3 Hidden Layers - Label vs. Predicted Label")
      plt.show()
```



```
[14]: plt.figure(figsize=(8, 8))
      colors = [int(x) for x in y_test]
      sc = plt.scatter(y_test, mlp_ts_pred[:, 3], s=1, c=colors,
                      cmap='twilight_shifted')
      plt.plot(y_test, y_test, color='dimgray', label='y_test=y_test', linewidth=0.7)
      plt.xlabel("y_ts")
      plt.ylabel("y_ts_pred_mlp4")
      plt.legend(*sc.legend_elements(), title='Azimuth')
      plt.title("MLP 4 Hidden Layers - Label vs. Predicted Label")
      plt.show()
```



[]: