1.1 Background

I picked the animal noise dataset because the ways animals behave is an extremely interesting field of study. As an animal rights advocate myself, I think it's very important to study behaviors like these in order to potentially identify stress, anxiety, and other maladaptive responses within these animals, so we can then try to find the cause of these stressors and make an effort to negate them. Regarding this dataset specifically, I thought the key problem was very interesting as well. Instead of just simply constructing the best model, the goal of this project was to find the key, or most important, features that could identify the three groups.

1.2 Data Description

The first thing I did was drop the variables SubjectID and Run because they were variables that didn't seem like they would contribute anything to the actual modeling process; they were simply labels used for describing the collection of the data. Next, I accounted for all NA values in the dataset by converting them all to 0's, as I realized that dropping all NA values by either row or column would be impractical as it would get rid of too high a proportion of data from the dataset. Then, I used Ordinal Encoding to encode the strengths of both the stimulus intensity level and the target value of level of exposure. I chose Ordinal Encoding over One Hot Encoding as the categorical values seemed to be numerically increasing in a line as compared to just being completely disparate from one another. Next, I plotted the ROC curves and found the AUC scores for all features with respect to the target, taking out the 91db data to make it a binary classification problem, as the problem stated that in particular, we wanted to find the difference between the control and the 96db group. The five features that had the best AUC scores were A_C1W1T2, A_C2W1T2, cC_C1W1T2, lC_C2W1T2, and cC_C2W1T2, which was definitely interesting as they all were of T2 (two weeks post noise exposure) and W1, a value that I unfortunately do not what it means. I decided to dive deeper and plot a heatmap of these 5 features and the target variable. What I found was that across the board, the heatmap affirms what I learned when I plotted the ROC Curves and calculated the AUC scores of all the feature variables. A_C1W1T2 has the highest correlation with the target, followed by A_C2W1T2, and then cC_C1W1T2, lC_C2W1T2, and cC_C2W1T2. However, I did take notice as to how correlated the feature variables were to each other as well. All of the correlations were above 0.5, with most correlation values even above 0.7. To me, this just shows that some of the best feature variables are accounting for the same things, and that is maybe why they are all so strong in the first place. After that, I scaled all my data between 0 and 1 using the MinMaxScaler from sklearn to ensure that no one features overpowered another when using them to fit and predict using a model. Then, I executed a train_test_split using the sklearn library.

1.3 Previous Work

I did not look at any previous implementations that used this dataset.

2. Methods

The four models I chose for this project were Logistic regression with L1 Penalty (Pedregosa, Varoquaux, et al.) Random Forest (Pedregosa, Varoquaux, et al.), Explainable Boosting Machine (Nori, Jenkins, et al.), and Gradient Boosting Classifier (Pedregosa, Varoquaux, et al.). I chose these with the goal of finding the best models that could also tell us useful information about variable importance. This is because the target of using this dataset as stated was "to identify key ABR metrics to distinguish these groups." So, while accuracy and good performance were important here, it is equally if not more so important to look into what variables are most significant within these models. So, I picked the first three models because of their ease of access to variable importance metrics and I picked GradientBoostingClassifier as a baseline, because while it is completely black-box and no sense of variable importance can generally be extracted from it, I wanted its performance to serve as sort of a benchmark for the other models I implement as it is generally known to give some of the best performances of standard datasets like this one, especially when it comes to Kaggle competitions.

Logistic regression minimizes log loss to optimize its parameter settings. Random Forest Classifier either minimizes the Gini loss or maximizes information gain. EBM minimizes the GA2M loss. Gradient Boosting Classifier can either use log loss or exponential loss to resemble more of an Adaboost classifier.

For this dataset, I generally used accuracy as my evaluation metric, as there is no case such that a false positive is considered to be worse than a false negative or vice versa, so there is no need for the use of alternative metrics like precision or recall.

3.1 Results

The Explainable Boosting Machine (EBM) and RandomForestClassifier performed the best with accuracies of approximately 0.838 over the test set. Then, the Logistic Regression with L1 penalty had an accuracy of approximately 0.811, and finally the GradientBoostingClassifier had an accuracy of approximately 0.784. Based on these results, it seems like EBM would be the best algorithm to use here because not only did it perform the best and have an easy way to determine variable importance from it, but it is also very interpretable, which causes it to slightly edge out the RandomForestClassifier, which has the first two qualities as well, but is considered to be relatively more black-box. The third best algorithm would be Logistic Regression with L1 loss, which does introduce the most sparsity, as can be seen from its graph, but does not have the same accuracy as the first two. The worst algorithm by far was GradientBoostingClassifier, which had the worst performance and is also the most black-box when it comes to notions of variable importance.

3.2 Hyperparameter Selection

For each of the 4 models, I tuned a different hyperparameter using KFold cross validation. For the logistic regression model with L1 penalty, I tuned the C value, or regularization strength and

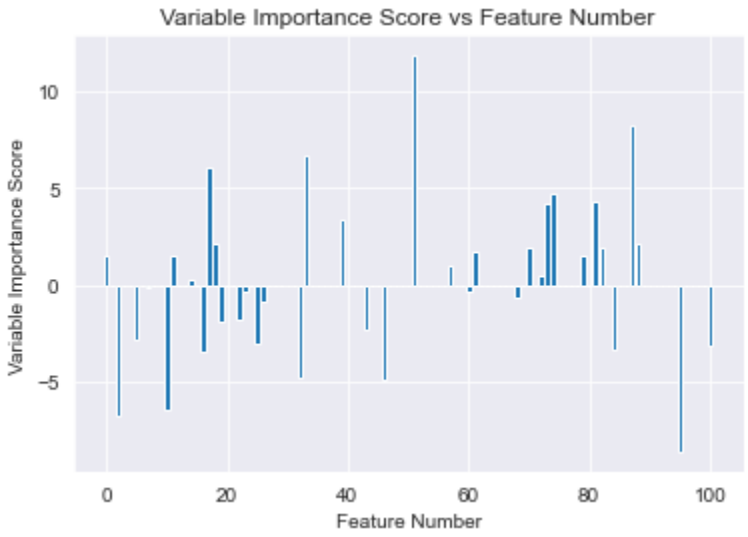
found that a value of 10 gave me the best accuracy. For the RandomForestClassifier, I tuned the minimum impurity decrease and found a value of 0.01 to be the best. For the ExplainableBoostingMachine (EBM), I tuned the learning rate and found a value of 0.01 to be the best. And finally, for the GradientBoostingClassifier, I tuned over the learning rate and found a value of 0.5 to be the best.

I have included plots for this process in my notebook. As I stated in my Data Description section, I executed a train_test_split using the sklearn library to break the dataset into training and test.

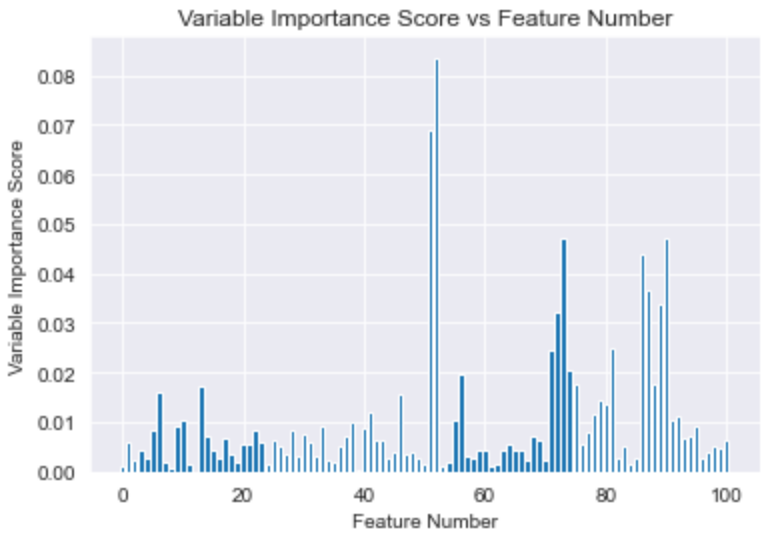
3.3 Visualizations

These are three visualizations of variable importance in the first three models I created.

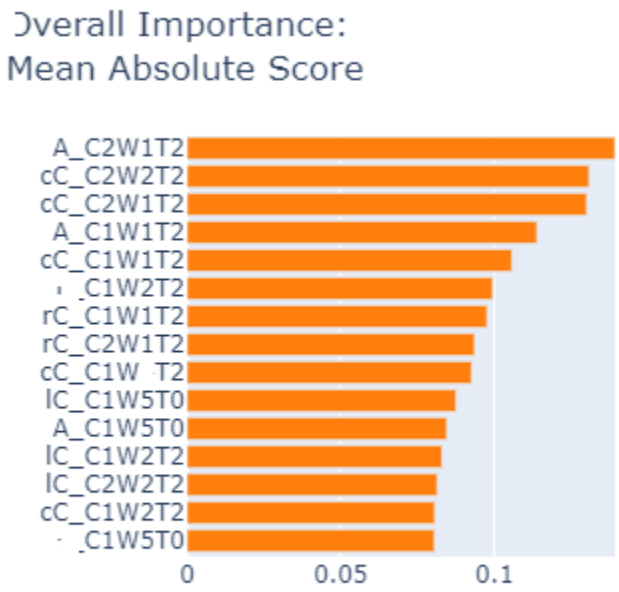
Logistic Regression:



RandomForestClassifier:



Explainable Boosting Machine:



3.4 Variable Importance

From my logistic regression model, I found that the five variables with highest importance, in order, were
A_C1W1T2, IC_C2W1T2, IC_C1W5T0, A_C1W5T0, IC_C1W1T2.

From my random forest classifier model, I found that the five variables with highest importance were
A_C2W1T2, A_C1W1T2, lC_C1W1T2, cC_C2W1T2, lC_C1W5T2

From my EBM model, I found that the five variables with highest importance, in order, were A_C2W1T2, cC_C2W2T2, cC_C2W1T2, A_C1W1T2, cC_C1W1T2.

All three of these models have A_C1W1T2 as one of their most important features, making it so that is most likely the most impactful feature in the dataset. The first two models also have lC_C1W1T2 as a shared important feature, while the second two models have A_C2W1T2 and cC_C2W1T2 as shared important features.

And bringing back my initial five features that had the best AUC scores were A_C1W1T2, A_C2W1T2, cC_C1W1T2, lC_C2W1T2, and cC_C2W1T2.

A_C1W1T2, which had the best AUC score, was also one of the most important features in all 3 of the models. For the rest of the four features with the best AUC scores, I have written them down along with what models they appear in.

cC_C1W1T2: 3

lC_C2W1T2: 1

A_C2W1T2: 1 and 2

cC_C2W1T2: 2 and 3

4. Insights

The features I found that were the most important tended to be of type T = 2, which intuitively makes sense because the value of the measurements two weeks post noise exposure deal a lot more directly with reaction of the animal to the said noise than at T0, which is pre noise exposure. These measurements of their reactions would give us a lot more information as to what reactions they are having and consequently, what stimuli they are reacting to.

Additionally, 3, 4, and 4 of the most important variables, respectively to the models, came with a value of W = 1. Unfortunately, based on the limited knowledge I have of this dataset, I do not know what a value of W = 1 means, but if someone was to look deeper into choosing the most important, or key, features to distinguish these three groups, that is definitely something I would advise them into looking into.

5. Errors

One misstep I may have made is that I did not consider the potential imbalanced nature of this dataset.

As for bugs, one bug that really slowed me down was the format my data was in. The amount of different formats that the inputs and outputs of different functions that I both implemented and imported were in (pandas dataframe, pandas series, numpy array, python list, etc.) got really confusing and often resulted in painful amounts of debugging to figure out where I was feeding in the wrong format of input.

Citations

Pedregosa, F., Varoquaux, Ga"el, Gramfort, A., Michel, V., Thirion, B., Grisel, O., … others. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*(Oct), 2825–2830.

Nori, H., Jenkins, S., Koch, P., & Caruana, R. (2019). Interpretml: A unified framework for machine learning interpretability. *arXiv preprint arXiv:1909.09223*.

Code

```
#!/usr/bin/env python
# coding: utf-8

# ### Loading in Libraries and Data

# In[427]:


import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt


# In[428]:


animaldf = pd.read_csv("FINAL Animal Data 2022.csv")


# In[429]:
```

```
animaldf


# ### Data Preprocessing

# #### Dropping SubjectID and Run

# In[430]:


animaldf = animaldf.drop(["SubjectID", "Run"], axis = 1)


# #### Accounting for NaN values

# ##### Seeing if dropping rows is viable to get rid of Nan values

# In[431]:


animaldf.shape[0]


# In[432]:


animaldf_test = animaldf.dropna()

animaldf_test.shape[0]


# Dropping rows is not a viable option because we will lose 76 out of 183
rows.

# ##### Seeing if dropping columns is viable to get rid of Nan values

# In[433]:


```

```
animaldf.shape[1]


# In[434]:


animaldf_test2 = animaldf.dropna(axis=1)

animaldf_test2.shape[1]


# Dropping columns is not a viable option because we will lose 38 out of
102 columns.

# ##### Filling in 0s for NaN values

# In[435]:


animaldf = animaldf.replace(np.nan, 0)


# Dropping all NA values by either row or column would be impractical as
it would get rid of too high a proportion of data from the dataset.

# #### Ordinal encoding for categorical data

# In[436]:


animaldf["Levels"].replace(["70_70", "70_90", "90_90"], [0, 1, 2], inplace
= True)


# In[437]:


animaldf["Group"].replace(["control", "91db", "96db"], [0, 1, 2], inplace
= True)
```

```
# #### ROC curves/AUC for just control vs 96 db group

# In[438]:


#plt.figure(figsize=(20,10))
#sns.heatmap(animaldf.corr(),vmin=-1,vmax=1,annot=True)


# In[439]:


animaldf_mod = animaldf[animaldf["Group"] != 1]


# In[440]:


X_mod = animaldf_mod.drop(["Group"], axis=1)
y_mod = animaldf_mod.iloc[:, 0]


# In[336]:


from sklearn.linear_model import LogisticRegression
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import roc_auc_score

ax = plt.gca()
auc_array = []

for i in range(X_mod.shape[1]):

    x_i = X_mod.iloc[:, i].to_numpy().reshape(-1, 1)
    y_i = y_mod.to_numpy()

    classifier = LogisticRegression()
    classifier.fit(x_i, y_i)
```

```
    rfc_disp = RocCurveDisplay.from_estimator(classifier, x_i, y_i, ax=ax,
alpha=0.8)

    auc_array.append(roc_auc_score(y_i,
classifier.decision_function(x_i)))

plt.legend(bbox_to_anchor=(1.1, 1.05))
plt.show()



# In[441]:


len(auc_array)


# In[442]:


valid_indices = []

tot = 0

for i in range(len(auc_array)):
    if auc_array[i] > 0.85:
        tot+=1
        valid_indices.append(i)

tot


# In[443]:


valid_indices


# In[444]:
```

```
cols = list(X_mod.columns)

for i in valid_indices:
    print(cols[i])


# These are the 5 features with the highest AUC scores.

# #### Creating Heatmap of Features with Highest AUC Scores and Target

# In[445]:


heatmapdf = animaldf[["Group", "A_C1W1T2", "A_C2W1T2", "cC_C1W1T2",
"lC_C2W1T2", "cC_C2W1T2"]]

plt.figure(figsize=(20,10))
sns.heatmap(heatmapdf.corr(),vmin=-1,vmax=1,annot=True)


# Across the board, the heatmap affirms what I learned when I plotted the
ROC Curves and calculated the AUC scores of all the feature variables.
A_C1W1T2 has the highest correlation with the target, followed by
A_C2W1T2, and then cC_C1W1T2, lC_C2W1T2, and cC_C2W1T2.
#
# However, something to note here is how correlated the feature variables
are to each other as well.  All of the correlations are above 0.5, with
most correlation values even above 0.7.  This just shows that some of the
best feature variables are accounting for the same things, and that is
maybe why they are all so strong in the first place.

# #### Popping out Target Variable

# In[246]:


y = animaldf.iloc[:, 0]

X = animaldf.drop(["Group"], axis=1)
```

```
# In[247]:


cols = list(X.columns)


# #### Scaling all values in X to values between 0 to 1 so none overpower
the others

# In[192]:


from sklearn.preprocessing import MinMaxScaler

scaler=MinMaxScaler()
scaled_X=pd.DataFrame(scaler.fit_transform(X), columns = cols)


# In[193]:


scaled_X


# #### Train Test Split

# In[194]:


from sklearn.model_selection import train_test_split

seed = 2022
X_train, X_test, y_train, y_test = train_test_split(scaled_X, y, test_size
= 0.2, random_state = seed)


# In[270]:
```

```
y_train = np.array(y_train)
y_test = np.array(y_test)


# ### Models

# #### Evaluation Metrics

# In[264]:


#cross validate over accuracy

def getAccuracy(actual, predicted):
    numCorrect = 0

    for i in range(len(actual)):
        if predicted[i] == actual[i]:
            numCorrect += 1

    return numCorrect/len(actual)



# #### Logistic Regression with L1 penalty

# In[458]:


from sklearn import linear_model

def runLog1(C_in):

    kf = KFold(n_splits = 5)

    sumAcc = 0

    for train_index, val_index in kf.split(X_train):
```

```
        X_train_only , X_val = X_train.iloc[train_index,:],
X_train.iloc[val_index,:]
        y_train_only , y_val = y_train[train_index], y_train[val_index]

        model = linear_model.LogisticRegression(penalty="l1",
solver="liblinear", C = C_in)

        model.fit(X_train_only, y_train_only)
        pred_values = model.predict(X_val)

        #print(type(pred_values))

        acc = getAccuracy(pred_values.tolist(), y_val.tolist())
        sumAcc += acc

    return sumAcc/5


# In[459]:


decrArray = [0.01, 0.1, 1, 10, 100]
outArray = []

for decr_val in decrArray:
    outArray.append(runLog1(decr_val))

outArray


# In[460]:


from math import log

log_c_vals = []

for val in decrArray:
    log_c_vals.append(round(log(val, 10), 2))
```

```
plt.plot(log_c_vals, outArray)

plt.title("Accuracy vs Log C Value")
plt.xlabel("Log C Value")
plt.ylabel("Accuracy")

plt.show()


# In[461]:


model1 = linear_model.LogisticRegression(penalty="l1", solver="liblinear",
C = 10)

model1.fit(X_train, y_train)


# #### Random Forest

# In[473]:


from sklearn.model_selection import KFold
from sklearn import ensemble

def runForest(decrease):

    kf = KFold(n_splits = 5)

    sumAcc = 0

    for train_index, val_index in kf.split(X_train):
        X_train_only , X_val = X_train.iloc[train_index,:],
X_train.iloc[val_index,:]
        y_train_only , y_val = y_train[train_index], y_train[val_index]

        model = ensemble.RandomForestClassifier(min_impurity_decrease =
decrease)
```

```
        model.fit(X_train_only, y_train_only)
        pred_values = model.predict(X_val)

        #print(type(pred_values))

        acc = getAccuracy(pred_values.tolist(), y_val.tolist())
        sumAcc += acc

    return sumAcc/5


# In[475]:


decrArray = [0.001, 0.005, 0.01, 0.05, 0.1]
outArray = []

for decr_val in decrArray:
    outArray.append(runForest(decr_val))

outArray


# In[477]:


from math import log

log_imp_vals = []

for val in decrArray:
    log_imp_vals.append(round(log(val, 10), 2))

plt.plot(log_imp_vals, outArray)

plt.title("Accuracy vs Log Min Impurity Value")
plt.xlabel("Log Min Impurity Value")
plt.ylabel("Accuracy")

plt.show()
```

```
# In[472]:


model2 = ensemble.RandomForestClassifier(min_impurity_decrease = 0.01)

model2.fit(X_train, y_train)


# #### Explainable Boosting Machine

# In[479]:


import interpret

from interpret import set_visualize_provider
from interpret.provider import InlineProvider
set_visualize_provider(InlineProvider())

import pandas as pd
from sklearn.model_selection import train_test_split

from interpret.glassbox import ExplainableBoostingClassifier
from interpret import show


# In[480]:


def runEBM(lr):

    kf = KFold(n_splits = 5)

    sumAcc = 0

    for train_index, val_index in kf.split(X_train):
        X_train_only , X_val = X_train.iloc[train_index,:],
X_train.iloc[val_index,:]
```

```
        y_train_only , y_val = y_train[train_index], y_train[val_index]

        model = ExplainableBoostingClassifier(learning_rate = lr)

        model.fit(X_train_only, y_train_only)
        pred_values = model.predict(X_val)

        #print(type(pred_values))

        acc = getAccuracy(pred_values.tolist(), y_val.tolist())
        sumAcc += acc

    return sumAcc/5


# In[481]:


decrArray = [0.001, 0.005, 0.01, 0.05, 0.1]
outArray = []

for decr_val in decrArray:
    outArray.append(runEBM(decr_val))

outArray


# In[482]:


from math import log

log_lr_vals = []

for val in decrArray:
    log_lr_vals.append(round(log(val, 10), 2))

plt.plot(log_lr_vals, outArray)

plt.title("Accuracy vs Log Learning Rate Value")
```

```
plt.xlabel("Log Learning Rate Value")
plt.ylabel("Accuracy")

plt.show()


# In[297]:


model3 = ExplainableBoostingClassifier(learning_rate = 0.01)

model3.fit(X_train, y_train)


# #### Gradient Boosting Classifier

# In[483]:


def runBoost(rate):

    kf = KFold(n_splits = 5)

    sumAcc = 0

    for train_index, val_index in kf.split(X_train):
        X_train_only , X_val = X_train.iloc[train_index,:],
X_train.iloc[val_index,:]
        y_train_only , y_val = y_train[train_index] , y_train[val_index]

        model = ensemble.GradientBoostingClassifier(learning_rate = rate)

        model.fit(X_train_only,y_train_only)
        pred_values = model.predict(X_val)

        acc = getAccuracy(pred_values.tolist(), y_val.tolist())
        sumAcc += acc

    return sumAcc/5
```

```
# In[484]:


rateArray = [0.001, 0.005, 0.01, 0.05, 0.1, 0.5]
outArray = []

for rate_val in rateArray:
    outArray.append(runBoost(rate_val))

outArray


# In[485]:


from math import log

log_lr_vals = []

for val in outArray:
    log_lr_vals.append(round(log(val, 10), 2))

plt.plot(log_lr_vals, outArray)

plt.title("Accuracy vs Log Learning Rate Value")
plt.xlabel("Log Learning Rate Value")
plt.ylabel("Accuracy")

plt.show()


# In[289]:


model4 = ensemble.GradientBoostingClassifier(learning_rate = 0.5)

model4.fit(X_train, y_train)
```

```
# ### Performances of Models on Test Set

# #### Logistic Regression with L1 penalty

# In[293]:


test_pred = model1.predict(X_test)

test_acc = getAccuracy(y_test.tolist(), test_pred.tolist())

test_acc


# #### Random Forest

# In[345]:


test_pred = model2.predict(X_test)

test_acc = getAccuracy(y_test.tolist(), test_pred.tolist())

test_acc


# #### Explainable Boosting Machine

# In[346]:


test_pred = model3.predict(X_test)

test_acc = getAccuracy(y_test.tolist(), test_pred.tolist())

test_acc


# #### Gradient Boosting Classifier
```

```
# In[299]:


test_pred = model4.predict(X_test)

test_acc = getAccuracy(y_test.tolist(), test_pred.tolist())

test_acc


# ### Variable Importance

# #### Function to find N max values from a list (used to find most
important variables)

# In[446]:


def Nmaxelements(list1, N):
    final_list = []
    index_list = []

    for i in range(0, N):
        max1 = 0
        index1 = 0

        for j in range(len(list1)):
            if (list1[j]) > max1:
                max1 = list1[j];
                index1 = j


        list1.remove(max1);
        final_list.append(max1)
        index_list.append(index1)


    return(final_list, index_list)
```

```
# #### Logistic Regression with L1 penalty

# In[447]:


importance=model1.coef_[0]
importance


# In[448]:


importance_abs = list(map(abs, importance))


dub, indices = Nmaxelements(importance.tolist(), 5)

dub


# In[449]:


indices


# In[450]:


for i in indices:
    print(cols[i])


# These are the 5 features with the highest variable importance in order.

# In[451]:


#plotting the features and their score
sns.set_style("darkgrid")
```

```
plt.bar([i for i in range (len(importance))],importance)

plt.title("Variable Importance Score vs Feature Number")
plt.xlabel("Feature Number")
plt.ylabel("Variable Importance Score")

plt.show()


# #### Random Forest

# In[452]:


importance = model2.feature_importances_
importance


# In[453]:


importance_abs = list(map(abs, importance))


dub, indices = Nmaxelements(importance.tolist(), 5)

dub


# In[454]:


indices


# In[455]:


for i in indices:
    print(cols[i])
```

```
# These are the 5 features with the highest variable importance in order.

# In[456]:


#plotting the features and their score
sns.set_style("darkgrid")
plt.bar([i for i in range (len(importance))],importance)

plt.title("Variable Importance Score vs Feature Number")
plt.xlabel("Feature Number")
plt.ylabel("Variable Importance Score")

plt.show()


# #### Explainable Boosting Machine

# In[311]:


#plotting the features and their score in descending order

ebm_global = model3.explain_global()
show(ebm_global)


# The five features with the highest variable importance, in order, are
A_C2W1T2, cC_C2W2T2, cC_C2W2T2, A_C1W1T2, and cC_C1W1T2.
```