Hunter Klein (Kaggle Username: mhunterklein)

Theory and Algorithms of ML

Rudin

11/18/20

Kaggle Project: AirBnB Dataset

1. Exploratory Analysis

This dataset provided a mix of continuous, categorical, and Boolean variables which needed to be parsed in their own way. Before being able to determine the utility of each variable type, categorical and text-based variables needed to be converted to a numerical format. Starting with the features, ‘last review’ and ‘host since’, I converted their date format, ‘mm/dd/yy’, features to number of days since last reviews and total number of days as a host given the current date. Next, I examined the categorical variables of ‘Room_Type’, ‘Bed_Type’, ‘Cancellation Policy’, and ‘Neighborhood’.

For Room_Type, possible values included Entire home/apt, private room, shared room, and hotel room. Entire home/apt and private room made up almost all the values as shared and hotel room only had 67 values between the two of them. To minimize the number of one-hot-encoded features, I only kept Entire home/apt as a feature for the final dataset. For Bed Type, regular bed made up all but 40 of the samples, so this feature would not be very helpful in predicting price, so I removed it from the dataset. For cancellation policy, I collapsed the 5 categories of Flexible, Moderate, Strict_14_day, strict_30_day, and strict_60_day into 3 categories of Flexible, Moderate, and Strict by combining all strict categories into one.

Neighborhood was by far the most tedious feature to work with as there were over 20 possible values and while some of them had hundreds or thousands of samples, over half of those possible values had less than 100 samples attributed to them. I ended up one-hot-encoding these and only keeping the ones that had a higher correlation coefficient than the ‘id’ marker.

Similarly, I wanted to avoid the noise generated by features that had very low correlation with the labels, so I only included features that had correlation coefficients higher than ‘id’ or 0.09. The top 5 of these remaining features were [‘Entire home/apt’, ’cleaning_fee’, ’bedrooms’, ’beds’, ’guests_included’].

I was hopeful to find some way to combine features, but no combination of two features was obvious that it would be helpful.

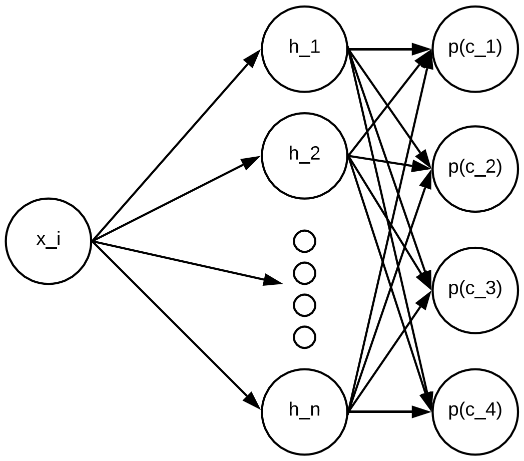
2. Splitting up the Data and Generalization

To avoid overfitting, I performed a stratified split on the training data over the price attribute to ensure equal proportions of each class were present in the training and validation split. 80% of the data went into the now true training data and 20% went to the validation dataset. The test set was left as a perfect hold out to test the models tuned on the training and validation sets only a single time. As I mention later, I was mistaken in thinking that the test set was something I could validate with at the end. If I hadn't made this mistake, I would've split the training set 70-20-10 into training, validation and test datasets. Validation to update parameters and avoid overfitting, and then test as a final sanity check before submission. I also would've made greater use of KFolds cross-validation.

3. Models

3.1 Generalized Additive Model (GAM)

For the first model approach, I chose to use a generalized additive model. Traditionally, GAMs are formulated as a univariate response function as $y = \beta_0 + f_1(x_1) + f_2(x_2) + \cdots + f_n(x_n)$ for a dataset $X \in \mathbb{R}^n$. Each function $f_1 \ldots f_n$ is independent used on a corresponding covariate $x_1 \ldots x_n$. To adapt to this task, I have made a few small adaptations to this model formulation to allow it to handle multiclass classification. First, I model each function f_i as a neural network using the PyTorch framework. Each function follows the same general multi-layer-perceptron architecture:



At the hidden layer, I also applied an ELU activation function. Finally, I push the sum of the networks through a softmax function. So, the final formulation of the model that I implemented is as follows. Where ϕ_i is the neural network approximation of f_i.

$$y = SoftMax(\beta_0 + \phi_1(x_1) + \phi_2(x_2) + \cdots + \phi_n(x_n))$$

My biggest draw to using this model was its interpretability. For a given prediction, we can examine the outputs of each approximating function ϕ_i to identify the influence of each variable x_i on prediction. If this dataset were used for a real business application to identify which airbnb's are a good investment – we'll want to know which factors influence that valuation the most. Additionally, a neural network approximation of each of these functions theoretically should be able to better capture non-linear relationships between the variables and the labels.

A major downside of this model is the computational complexity to train it – especially as the number of features goes up. After filtering for features that are linearly correlated with the target labels, I had less than 20 features and this was still taking too long to train at 40 minutes for 40,000 iterations using GPU acceleration. I ended up sub-selecting the 5 most correlated features and training the model off of those to allow for faster tuning of hyperparameters.

3.2 Gradient Boosted Classifier (Tree) - Sklearn

I selected gradient boosted decision trees as my second algorithm. This was motivated by the interesting and impressive performance that we observed during the class homework about Adaboost. Some of the components of gradient boosted decision trees that I liked most was their conceptual simplicity, and resilience to overfitting, strength of predictive power, and availability of good premade libraries. The library I chose to use was Sklearn's sklearn.enseble.GradientBoostingClassifier which unfortunately does not cite specific authors for the code. Sklearn is a ubiquitous machine learning library that contains implementations of many common learning algorithms that integrate easily with numpy and scipy objects. More details on this implementation will be shared during the training section.

In general, using gradient boosted decision trees was much less computationally complex than the neural network GAM that I used as my first network with a training time of less than a minute for our training dataset of 9000 samples.

4 Training

4.1 GAM

For the GAM network, training was performed using torch.optim.AdamW which is an implementation of the Adam optimization algorithm with built in weight decay. The Adam algorithm is similar to other stochastic gradient descent methods but provides for bias correction by using first and second moment estimation. In general the algorithm calculates the gradient of the loss, g, then using a set of exponential decay rate hyperparameters ρ_1, ρ_2, the model calculates the first and second moment using the following formulas:

$$\rho_1 s_t + (1-\rho_1)g \to s_{t+1}, \quad \rho_2 r_t + (1-\rho_2)g \odot g \to r_{t+1}$$

Bias is then corrected in these estimates by dividing by $(1-\rho^t)$ respectively. The parameters of the model are then updated using the following update rule and learning rate ϵ and a small numerical stabilization constant δ:

$$\Theta_{t+1} = \Theta_t - \epsilon\left(\frac{\hat{s}}{\sqrt{\hat{r}} + \delta}\right)$$

Now as mentioned, AdamW uses an implementation of decoupled weight decay outlined in (Loshchilov et al. 2019, https://arxiv.org/pdf/1711.05101.pdf). Which adds a decaying regularization term λ to the update step where:

$$\Theta_{t+1} = \Theta_t - \epsilon\left(\frac{\hat{s}}{\sqrt{\hat{r}} + \delta} + \lambda\Theta_t\right)$$

As mentioned in the previous section runtime was about 40 minutes for 40000 iterations while using GPU acceleration.

4.2 Gradient Boosted Decision Tree

In class we discussed coordinate descent for Adaboost where for each iteration, a set number of small permutations were made and tested to see which reduced the loss the best and then weights of the boosted tree were updated to match that incremental step. Gradient boosted works very similarly but where the new estimators are derived based on the gradient of some differentiable loss function.

In the case of Sklearn's implementation, the boosted decision tree is modeled as $F_M(x_i) = \sum_m h_m(x_i)$ where each function h_m is a weak estimator, or weighted decision tree. Now, sklearn has implemented their decision trees to always be regression decision trees that will output values in $\mathbb{R}$, so in order to yield a classification value / probability, they use $F_M(x_i)$ as input to the logistic function where $p(y_i = 1|x_i) = \sigma(F_M(x_i))$. In the case of multiclass classification K trees are built (for K classes) where $p(y_i = k|x_i) = SoftMax\left(F_{M,k}(x_i)\right)$.

For each new weak estimator added, it will be selected such that $h_m \approx \arg\min_h \sum_{i=1}^{n} h(x_i) g_i$ where g_i is the derivative of the loss function with respect to $F_{M,k}$. Since we are using the logistic function, the gradient is easily estimated at $\frac{d}{dF_M}\left[\sigma(F_M(x_i))\right] = \sigma(F_M(x_i))\left(1 - \sigma(F_M(x_i))\right)$. Thus, we can see that h_m will be selected to fit the negative gradient of the previous estimator F_{M-1} and $F_m(x) = F_{m-1}(x) + \lambda h_m(x)$ where λ is a learning rate parameter.

The maximum depth, maximum number of leaves, and learning rate are tunable hyperparameters. There also is a sampling hyperparameter to make the model stochastically update. Lastly, the model trains in less than a minute for the training data.

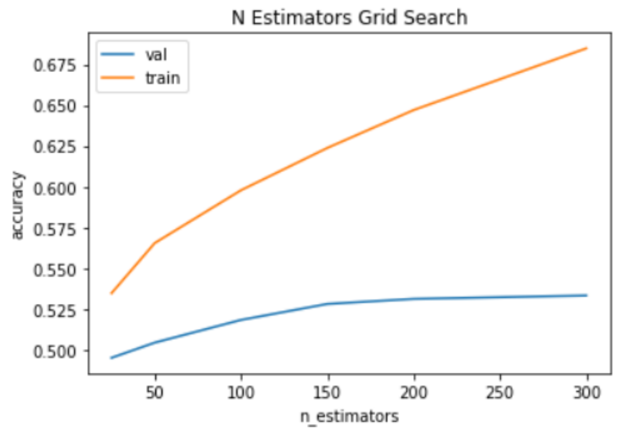
5 Hyperparameter selection

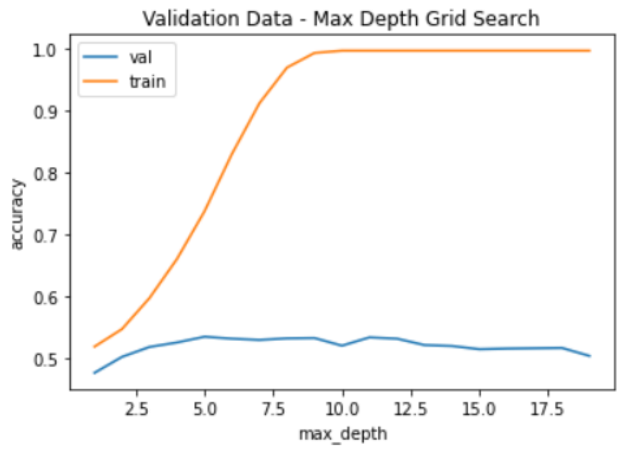
5.1 GAM

Given the long training time I trained this model using manual tuning and checking training / validation data performance. The primary hyperparameters I tuned were learning rate, number of training iterations and the size of the hidden layer. For learning rate I tried [1e-2,1e-3,1e-4,1e-5] where there was no significant increase in performance once the learning rate was smaller than 1e-3. For n_hidden I checked values of 12, 25, 50, 100, and 300 where values less than 50 and above 150 couldn't seem to optimize and reverted to random guessing. I also tried using different losses to see if my poor performance with this model was just due to poorly formulating the model. I tried using NLLLoss() and BCELoss() which I found out later are actually identical except BCELoss() has a softmax layer built into it. For training iterations, I tried 1000, 4000, 10000, 40000, and 100000 since sometimes more complicated neural networks just need more time to train. Regrettably, no additional performance was able to be achieved beyond 40000 iterations.

5.2 GradientBoostedDecisionTree

For Gradient Boosted Decision Tree, I used sklearn’s gridsearch CV to try out lots of different max_depth, n_estimators, learning_rates and both loss functions. The performance comparisons for N estimators and max depth vs accuracy are shown below using the training and validation sets. I ended up settling on n_estimators=150 and max_depth=6 as both of these values had near peak validation performance without the overfitting of the training data going to far. For learning rate I tested the same range as the GAM with no perceived improvement and for the loss functions, I tested both the discrepancy (logistic function) and exponential loss. The performance did not notably improve with the exponential loss.





6 Errors and Mistakes

6.1 GAM

I think selecting this model formulation in general was a mistake. I realized too late that the GAM formulation won’t capture the covariance between features well since it seems to assume that they are all quasi-independent. I think this is the reason that I was only able to achieve accuracy in the low 30’s. There’s enough information to do better than random guessing, but I would assume that there’s more information contained in the interaction of the features that the GAM missed out on.

Something else that I caught was that the GAM would only ever converge to predicting two of the 4 classes. I thought that perhaps the model just couldn't handle multi-class classification well. If I had a couple more days to work on the project, I would rewrite the model to have a different GAM for each class in a one-vs-all formulation similar to what happens in sklearn's boosted decision tree. Using the GAM as a weak predictor in Adaboost would also be interesting.

Fundamentally, I think there' s a lot I could've improved with the GAM and I may play around with it a bit over the break.

6.2 Gradient Boosted Decision Tree

For GBDT, I think things went well overall. There are some additional hyperparameters I would test moving forward that are more relevant to the construction of the estimators like max_leafs, stochastic sampling, and branch purity constraints. Otherwise I think that the GBDT went well and I'm not sure what else I would've changed by intuition.

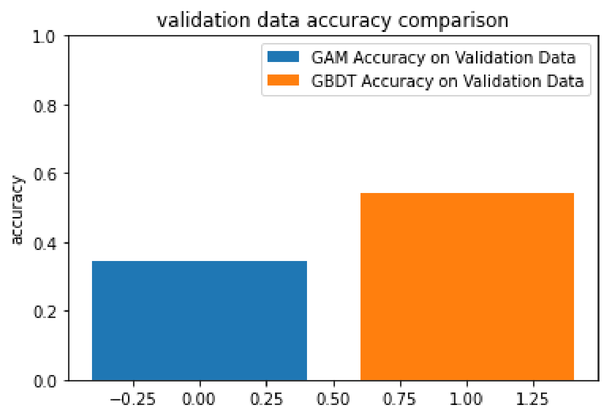
6.3 Data Processing Feature Engineering

Big one here – I didn't realize that test.csv was the true test set without labels, but rather a partitioned test set for training. So one thing that I made a mistake on was not splitting the training data into training, validation, *and test* sets – so that way I could have an untouched test set that I could actually validate my models on. Instead I was left to use the validation set over and over and it probably snuck some bias into my model selection. While I know I could've gone back to split another partition, I didn't realize my mistake until the end of the project and by that point I figured whatever bias I had in formulating my models was already there.

I also think it was a mistake to sub-select my features by a linear correlation coefficient. While I thought this would be a good metric, it leaves out any potential non-linear relationships – especially if it was a non-linear relationship between the covariance of some pair of features and the labels where either one or both items in the pair of features had poor linear correlation.

7 Accuracy

GAM Validation Accuracy: 0.346, GBDT Validation Accuracy: 0.543



Data Preprocessing

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import sklearn
from sklearn.model_selection import StratifiedShuffleSplit
import datetime
%matplotlib inline
```

Data Processing: Helper Functions

In [159]:
```
def mdy_to_days_since(df,feature_list,reference_day=datetime.date.today()):
    '''
    Description:
    This function converts mm/dd/yy data into a number of days from a certain
 reference date.

    Inputs
    -------------------
    df: Pandas dataframe
    feature_list: list of features which have dates in the format of mm/dd/yy
    reference_day: day to which you would like to calculate the difference in
 days

    Outputs
    -------------------
    copy: Pandas dataframe containing only information for days between dates
 and the reference day for each feature
    in the feature list

    '''
    copy = df[feature_list].copy()

    for feature in feature_list:
        for idx,value in enumerate(df[feature]):
            split_date = value.split('/')

            if int(split_date[2]) > 22:
                year = int(str(19) + split_date[2])
            else:
                year = int(str(20) + split_date[2])

            sample_day = datetime.date(month=int(split_date[0]),day=int(split_
date[1]),year=year)

            num_days_past = (reference_day - sample_day).days

            copy[feature][idx] = num_days_past

    return copy

def t_to_True(df,feature_list):
    '''
    Description:
    Converts the t and f char values to True and False boolean values in a sim
ilar manner to the mm/dd/yy function above
    '''
    copy = df[feature_list].copy()

    for feature in feature_list:
        for idx,value in enumerate(df[feature]):
            if value == 't':
                change = True
            else:
                change = False

            copy[feature][idx] = change
    return copy
```

Data Processing: Loading the Training Data

In [165]:
```
Training_Preprocessing = False
data = pd.read_csv('test.csv')
data.head()
```

Out[165]:

	id	neighbourhood	room_type	minimum_nights	number_of_reviews	last_review	reviews_
0	7715	San Nicolás	Entire home/apt	1	29	1/20/20	
1	13196	Recoleta	Entire home/apt	2	6	3/11/20	
2	13194	Palermo	Entire home/apt	20	3	3/15/20	
3	4673	Villa Crespo	Entire home/apt	20	69	3/15/20	
4	11325	Villa Urquiza	Entire home/apt	2	7	4/26/20	

5 rows × 24 columns

In [166]:
```
discrete_variables = ['neighbourhood','room_type','last_review','host_since',
'host_is_superhost','bed_type',
                     'instant_bookable','is_business_travel_ready','cancellati
on_policy','require_guest_profile_picture',
                     'require_guest_phone_verification']

categorical_variables= ['neighbourhood','room_type','bed_type','cancellation_p
olicy']

boolian_variables = ['host_is_superhost','instant_bookable',
                     'is_business_travel_ready','require_guest_profile_pictur
e',
                     'require_guest_phone_verification']

date_format_variables = ['last_review','host_since']

#variables with very very little to no predictive information
delete_variables = ['is_business_travel_ready','bed_type']

#Convert mm/dd/yy variables to days since event
discrete_days = mdy_to_days_since(data,date_format_variables)
data[date_format_variables] = discrete_days[date_format_variables].astype(np.i
nt64)

#Convert character 't' 'f' values to Boolean 1's and 0's
data[boolian_variables] = t_to_True(data,boolian_variables).astype(int)

#----------------------------
#Convert Categorical data to 1-hot vectors
#----------------------------
#Cancelation Policy
dummy_cancel_policy = pd.get_dummies(data['cancellation_policy'])

if Training_Preprocessing==True:
    strict_cancelation = dummy_cancel_policy['strict_14_with_grace_period'] +
dummy_cancel_policy['super_strict_30'] + dummy_cancel_policy['super_strict_60'
]
    strict_cancelation = pd.DataFrame(strict_cancelation, columns=['strict'])
    del dummy_cancel_policy['strict_14_with_grace_period']
    del dummy_cancel_policy['super_strict_30']
    del dummy_cancel_policy['super_strict_60']
else:
    strict_cancelation = dummy_cancel_policy['strict_14_with_grace_period'] +
dummy_cancel_policy['super_strict_30']
    strict_cancelation = pd.DataFrame(strict_cancelation, columns=['strict'])
    del dummy_cancel_policy['strict_14_with_grace_period']
    del dummy_cancel_policy['super_strict_30']
dummy_cancel_policy = dummy_cancel_policy.join(strict_cancelation)
del data['cancellation_policy']
data = data.join(dummy_cancel_policy)

#Room Type
dummy_room_type = pd.get_dummies(data['room_type'])
home_apartment = pd.DataFrame(dummy_room_type['Entire home/apt'])
home_apartment.rename(columns={'Entire home/apt':'HomeApt'})
del data['room_type']
```

```
data = data.join(home_apartment)


#Neighborhood
dummy_neighbourhood = pd.get_dummies(data['neighbourhood'])
del data['neighbourhood']
data = data.join(dummy_neighbourhood)
data.head()

#------------------------
#Delete unpredictive features
#------------------------
del data['is_business_travel_ready']
del data['bed_type']
```

In [167]:

```
if Training_Preprocessing==True:
    pd.set_option('max_rows',100)
    print(np.abs(data.corr()['price']).sort_values(ascending=False))
```

In [170]:

```
#Features with higher correlation coefficients than ID
if Training_Preprocessing==True:
    Top_Corr_Categories = ['id','Entire home/apt','cleaning_fee','bedrooms','b
eds','guests_included','bathrooms','Palermo','host_since',
                          'availability_365','calculated_host_listings_count',
'Almagro','strict','Puerto Madero','Balvanera',
                          'flexible','extra_people','maximum_nights','host_is_
superhost','price']
else:
    Top_Corr_Categories = ['id','Entire home/apt','cleaning_fee','bedrooms','b
eds','guests_included','bathrooms','Palermo','host_since',
                      'availability_365','calculated_host_listings_count','Alm
agro','strict','Puerto Madero','Balvanera',
                      'flexible','extra_people','maximum_nights','host_is_supe
rhost']

#Features to scale between 0 and 1
features_to_normalize = ['cleaning_fee','bedrooms','beds','guests_included','b
athrooms','host_since','availability_365',
                        'calculated_host_listings_count','extra_people','maxim
um_nights']

for feature in features_to_normalize:
    data[feature] = (data[feature] - data[feature].min()) / (data[feature].max
() - data[feature].min())
```

In [171]:
```
from sklearn.model_selection import StratifiedShuffleSplit

#Make the high correlation features dataframe
data_high_corr = data[Top_Corr_Categories].copy()

#stratified split the training data into training and validation sets along th
e price attribute
if Training_Preprocessing==True:
    split = StratifiedShuffleSplit(n_splits=1,test_size=0.2,random_state=27)
    for train_idx,test_idx in split.split(data_high_corr,data_high_corr['pric
e']):
        data_high_corr_train = data_high_corr.loc[train_idx]
        data_high_corr_val = data_high_corr.loc[test_idx]

    data_high_corr_train.to_csv('train_data_preprocessed.csv')
    data_high_corr_val.to_csv('val_data_preprocessed.csv')
else:
    data_high_corr.to_csv('test_data_preprocessed.csv')
```

Models

- 1: Generalized Additive Model (GAM) using DNNs as approximating functions (Self Implemented)
- 2: Gradient Boosted Decision Trees (Sklearn)

In [172]:
```
data_train = pd.read_csv('train_data_preprocessed.csv')
data_val = pd.read_csv('val_data_preprocessed.csv')
data_test = pd.read_csv('test_data_preprocessed.csv')

y_train = data_train['price'].astype(np.int64) - 1
X_train = data_train.drop(columns=['id','price','Unnamed: 0']).astype(np.float
64)

y_val = data_val['price'].astype(np.int64) - 1
X_val = data_val.drop(columns=['id','price','Unnamed: 0']).astype(np.float64)

X_test = data_test.drop(columns=['Unnamed: 0']).astype(np.float64)
```

Generalized Additive Model

I chose to use a generalized additive model using shallow(ish) neural networks to model f_1 ... f_n. The neural network approximation of the different univariate functions allows for some flexibility in modeling interesting relationships, while the additive sum of these functions provides an interpretable view of the importance of each factor.

$$y = Softmax(\beta_0 + f_1(x_1) + f_2(x_2) + \ldots + f_n(x_n))$$

In [10]:
```
import torch
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import torch.nn.functional as F

if torch.cuda.is_available():
    device = torch.device("cuda:0")  # you can continue going on here, like cu
da:1 cuda:2....etc.
    print("Running on the GPU")
else:
    device = torch.device("cpu")
    print("Running on the CPU")
```

```
Running on the GPU
```

In [26]:
```
class GAM(nn.Module):
    def __init__(self,n_features,n_classes,n_hidden):
        super(GAM,self).__init__()

        self.n_features = n_features
        self.n_classes = n_classes
        self.n_hidden = n_hidden
        self.function_list = nn.ModuleList()

        for i in range(n_features):
            self.function_list.append(nn.Sequential(
                nn.Linear(1,n_hidden),
                nn.ELU(),
                nn.Linear(n_hidden,n_classes),
            ))

        self.bias = nn.Parameter(torch.rand(n_classes),requires_grad=True)

    def forward(self,X):
        y = 0

        for feature in range(self.n_features):
            y = self.function_list[feature](X[:,feature].unsqueeze(1))
        y = y + self.bias
        y = F.softmax(y,dim=1)
        return y
```

In [27]:
```
features_to_care = ['Entire home/apt','cleaning_fee','bedrooms','beds','guests
_included','bathrooms']
N_FEATURES = X_train[features_to_care].shape[1]
N_CLASSES=4
N_HIDDEN=100
N_EPOCHS = 40000
BATCH_SIZE=5
LOSS = nn.NLLLoss()
#LOSS = nn.BCELoss()
print('hello')
model = GAM(n_features=N_FEATURES,n_classes=N_CLASSES,n_hidden=N_HIDDEN).to(de
vice)
print('world')
optimizer = optim.AdamW(model.parameters(),lr=1e-4)
```

```
hello
world
```

In [28]:
```
def batcher(X,batch_size=20):
    idx = np.random.choice(X.shape[0],size=batch_size,replace=False)
    return idx

loss_hist = []
for e in range(N_EPOCHS):
    optimizer.zero_grad()

    batch_idxs = batcher(X_train,batch_size=BATCH_SIZE)

    X_train_batch = torch.Tensor(X_train[features_to_care].to_numpy(dtype=np.f
loat64)[batch_idxs]).to(device)
    y_train_batch = torch.Tensor(y_train.to_numpy(dtype=np.float64)[batch_idxs
]).type(torch.LongTensor).to(device)

    y_pred = model.forward(X_train_batch)
    #print(y_pred)
    CE_loss = LOSS(y_pred,y_train_batch)

    CE_loss.backward()
    optimizer.step()

    loss_hist.append(CE_loss.detach().to('cpu').numpy())
```

In [31]:
```
y_pred_val_GAM = model.forward(torch.Tensor(X_val[features_to_care].to_numpy(d
type=np.float64)).to(device))
y_pred_val_GAM = np.argmax(y_pred_val_GAM.detach().to('cpu').numpy(),axis=1)
GAM_val_accuracy = np.sum(y_pred_val == y_val) / y_val.shape[0]
print("GAM validation Accuracy: ",GAM_val_accuracy)
```

```
GAM validation Accuracy:  0.3464119772844605
```

In [32]:
```
y_pred_test_GAM = model.forward(torch.Tensor(X_test[features_to_care].to_numpy
(dtype=np.float64)).to(device))
y_pred_test_GAM = np.argmax(y_pred_test_GAM.detach().to('cpu').numpy(),axis=1)
```

Gradient Boosted Decision Trees

In [122]:
```
import sklearn
from sklearn.ensemble import GradientBoostingClassifier as GBC
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
import pandas as pd
import numpy as np
```

In [173]:
```
CV_parameters = {
    'max_depth':(3,4,5,6,7),
    'n_estimators':(50,100,150)
}
model = GBC()
clf = GridSearchCV(model,CV_parameters)
clf.fit(X_train,y_train)
```

Out[173]:
```
GridSearchCV(estimator=GradientBoostingClassifier(),
             param_grid={'max_depth': (3, 4, 5, 6, 7),
                         'n_estimators': (50, 100, 150)})
```

In [203]:
```
clf.best_estimator_
```

Out[203]:
```
GradientBoostingClassifier(max_depth=5, n_estimators=150)
```

In [175]:
```
clf.best_score_
```

Out[175]:
```
0.540292144802932
```

In [176]:
```
CV_parameters = {
    'max_depth':[5],
    'n_estimators':(150,200,250)
}
clf = GridSearchCV(model,CV_parameters)
clf.fit(X_train,y_train)
```

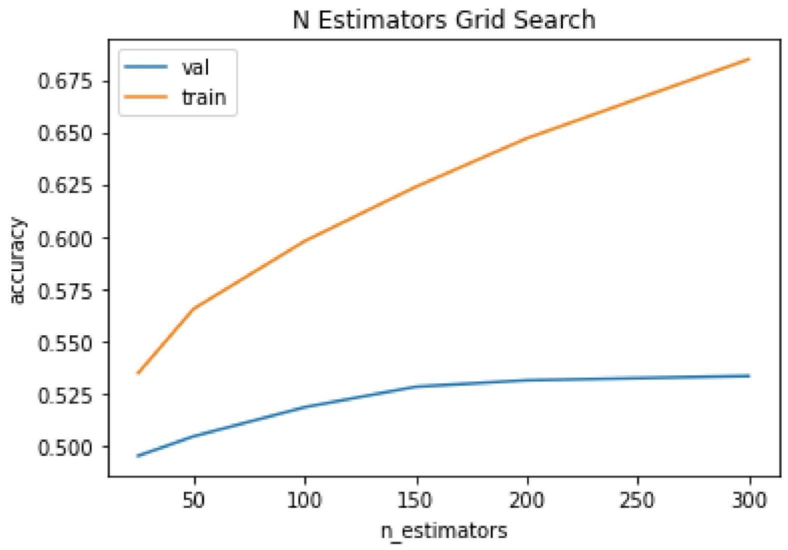
Out[176]:
```
GridSearchCV(estimator=GradientBoostingClassifier(),
             param_grid={'max_depth': [5], 'n_estimators': (150, 200, 250)})
```

In [209]:
```
score_hist = []
score_train_hist = []
for depth in [25,50,100,150,200,300]:
    print(depth)
    model = GBC(n_estimators=depth)
    model.fit(X_train,y_train)
    score_train_hist.append(model.score(X_train,y_train))
    score_hist.append(model.score(X_val,y_val))

plt.plot([25,50,100,150,200,300],score_hist,label='val')
plt.plot([25,50,100,150,200,300],score_train_hist,label='train')
plt.legend()
plt.xlabel('n_estimators')
plt.ylabel('accuracy')
plt.title('N Estimators Grid Search')
plt.plot()
```

```
25
50
100
150
200
300
```

Out[209]:
```
[]
```

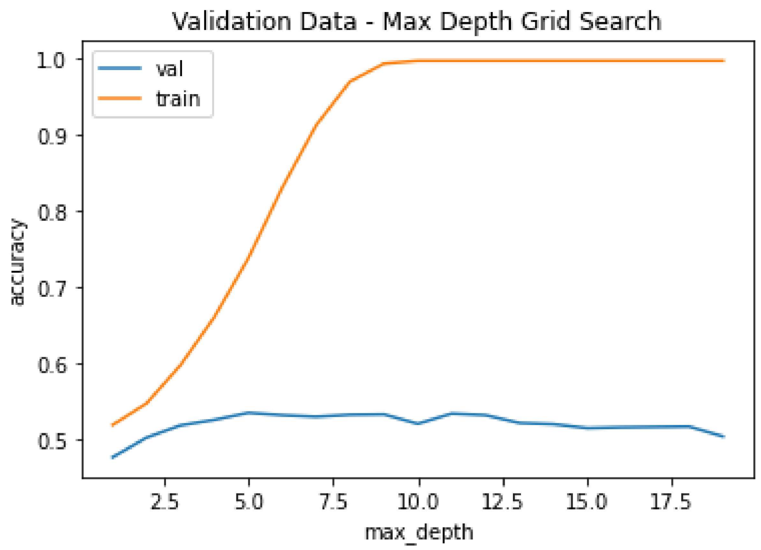


In [207]:
```
score_hist = []
score_train_hist = []
for depth in range(1,20):
    print(depth)
    model = GBC(max_depth=depth)
    model.fit(X_train,y_train)
    score_train_hist.append(model.score(X_train,y_train))
    score_hist.append(model.score(X_val,y_val))

plt.plot(range(1,20),score_hist,label='val')
plt.plot(range(1,20),score_train_hist,label='train')
plt.legend()
plt.xlabel('max_depth')
plt.ylabel('accuracy')
plt.title('Validation Data - Max Depth Grid Search')
plt.plot()
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
```

Out[207]:
```
[]
```



In [190]:
```
#Final Model
model = GBC(max_depth=6,n_estimators=150)
model.fit(X_train,y_train)
y_pred_val = model.predict(X_val)
acc_val = np.sum(y_pred_val == y_val) / y_val.shape[0]
print(acc_val)
```

```
0.5436241610738255
```

In [191]:
```
model.score(X_train,y_train)
```

Out[191]:
```
0.8916580578512396
```

In [192]:
```
y_pred_test_GBC = model.predict(X_test.drop(columns='id')) + 1
id_column = X_test['id'].copy().astype(int)
```

In [193]:
```
y_pred_test_GBC = pd.DataFrame({
    'id':id_column,
    'price':y_pred_test_GBC
})
```

In [194]:
```
y_pred_test_GBC.to_csv('y_pred_test_GBC.csv')
```

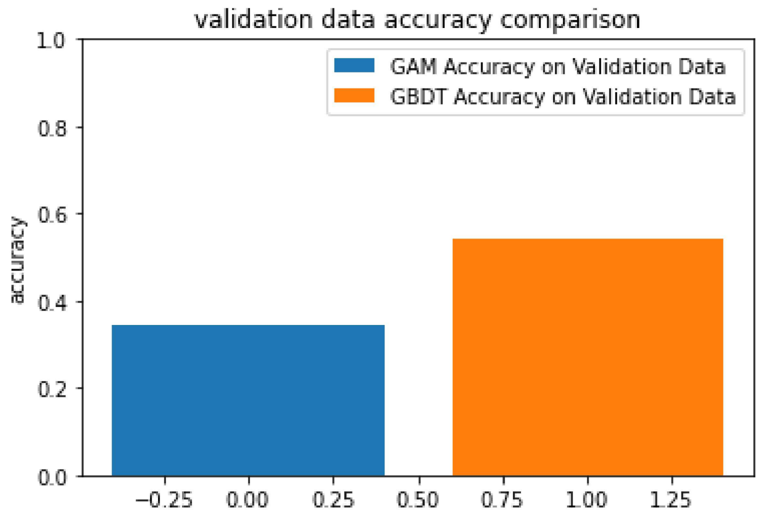
In [195]:
```
y_pred_test_GBC
```

Out[195]:

	id	price
0	7715	2
1	13196	3
2	13194	2
3	4673	3
4	11325	1
...	...	...
4144	12921	2
4145	7174	2
4146	9240	3
4147	11663	1
4148	4513	2

4149 rows × 2 columns

In [213]:
```
plt.bar([0],[0.346],label='GAM Accuracy on Validation Data')
plt.bar([1],[0.543],label='GBDT Accuracy on Validation Data')
plt.legend()
plt.ylim([0,1])
plt.ylabel('accuracy')
plt.title('validation data accuracy comparison')
plt.show()
```



In [ ]:
```
```