Machine Learning Final

Matthew Radtke

January 4, 2023

(i) (a) In this report, I will address the feasibility of predicting the Rating, Accuracy, Cleanliness, Checkin, Communication, Location, and Value scores of Air Bnb based on the reviews and listing data provided by Air Bnb on their website.

Once the data was retrieved, the first step was cleaning the data into usable numeric features to be used for linear regression. I created a C project to deserialize the data and perform filtering, compute new fields, and convert existing ones while dropping "dead" fields that did not contribute any value to the regression. These dead fields were fields such as Host Id, Listing Id, etc. I created new fields that were computed based on an existing fields. All true false fields were computed to be a -1 or +1 based on the value.

Such computed fields include:

host verifications-> Broken into many True/False fields for each type of verification method such as phone.

first review-> Compute years since first review.

last review-> Compute years since last review.

Once the new fields were computed, we needed to clean the description fields that allowed text such as drescription and neighborhood overview as well as the review comments. This involved removing all html tags from the text and then selecting only the alphanumeric characters to remain in the text. These cleaned text fields were not directly used in the regression. I first used the Affin sentiment analysis python package to get the sentiment value of all the text data. This meant that instead of the description being used, I computed the sentiment of the description and added that as a feature.

Reviews were marked separately from the listing data, but to incorporate them into the final data to be used for regression I computed the sentiment of all the reviews for a listing and added them together as a feature. This could potentially result in some skewed results though as some listings have a lot of reviews while some have few, so I also added the average of all review sentiments for a listing as a feature. This allowed me to be able to use the review data itself in some way instead of discarding it. The Affin package was used strictly in english to determine the sentiment of the reviews, so any sentences it did not recognize or in another language received a score of 0.

Other fields that contained lists of strings representing that data such as amenities were dropped from the final cleaned data.

The final feature list is as follows:

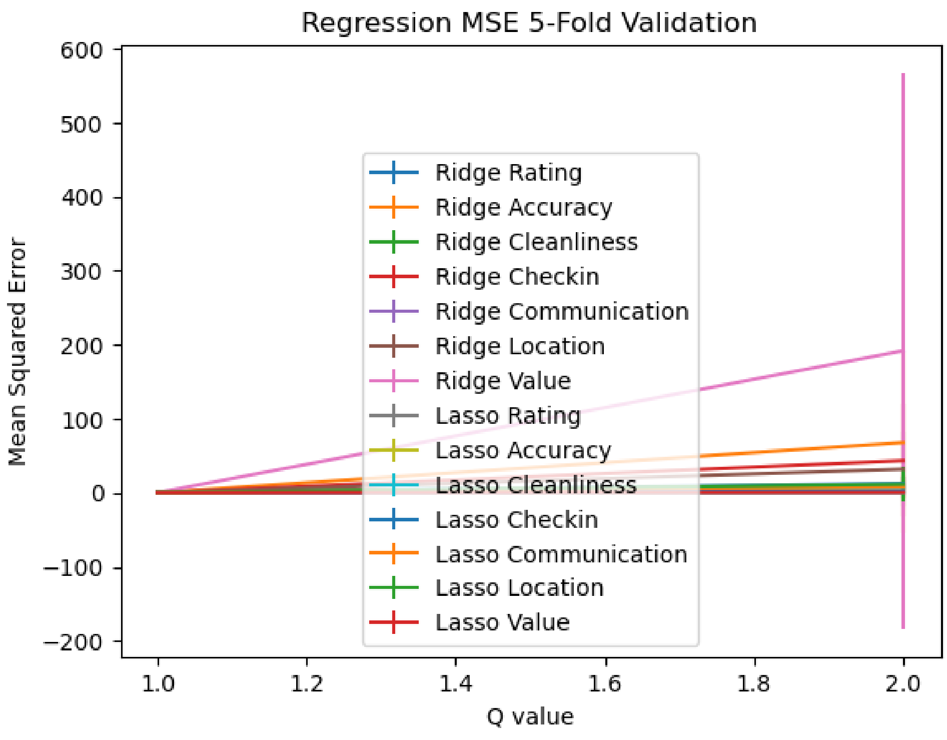
name_sentiment
description_sentiment
neighborhood_overview_sentiment
host_duration_years
host_about_sentiment
host_response_rate
host_acceptance_rate
host_is_superhost
host_listings_count
host_total_listings_count
host_phone_verified
host_email_verified
host_work_email_verified
host_has_profile_pic
host_identity_verified
accommodates
bathrooms
bedrooms
beds
price
minimum_nights
maximum_nights
minimum_minimum_nights
maximum_minimum_nights
minimum_maximum_nights
maximum_maximum_nights
minimum_nights_avg_ntm
maximum_nights_avg_ntm
has_availability
availability_30
availability_60
availability_90
availability_365
number_of_reviews
number_of_reviews_ltm
number_of_reviews_l30d
years_since_first_review
years_since_last_review
instant_bookable
calculated_host_listings_count
calculated_host_listings_count_entire_homes
calculated_host_listings_count_private_rooms
calculated_host_listings_count_shared_rooms
reviews_per_month
review_sentiment
review_sentiment_average

Not all listings had reviews, so I removed the listings that did not have any review data associated with them. Once the data was cleaned, I created separate files for each score we are trying to predict and where there is data for that score in the first column. So even though there may be a checkin score for a listing, there may not be a cleanliness score for that same listing. Most of the data had all of their respective scores, but I removed those without a score for their respective file. In other words, if the cleanliness file had missing cleanliness scores for that listing, I removed that listing from the dataset.

With the feature selection and data cleaning complete, I chose to evaluate both Lasso and Ridge regression to see if it is possible to predict the air bnb scores. The key difference between Ridge and Lasso regression is the penalty associated with each. The L1 penalty associated with Lasso regression seeks to make as many components of $\theta = 0$ as possible. The L2 penalty does not make the weights 0, but instead tries to have each feature contribute a bit to the linear model. It tends to make the weights very small instead of 0.

The dataset was divided into 2 randomly shuffled pieces, training (90%) and validation (10%). The cross validation and training happened all on the training portion of the data. The metric I used for evaluation was Mean Squared Error (MSE). The features were all in the same order of magnitude relatively speaking, so no feature scaling was required to be applied to the features.

The first steps were to choose the ideal hyper parameters for each of the models for each of the scores we are trying to predict. This involved first deciding the ideal Q value for polynomial features of the data set. With a default C = 1 value, I ran cross validation to choose Q. Running 5 fold cross validation on the training portion of the data gave the following figure:

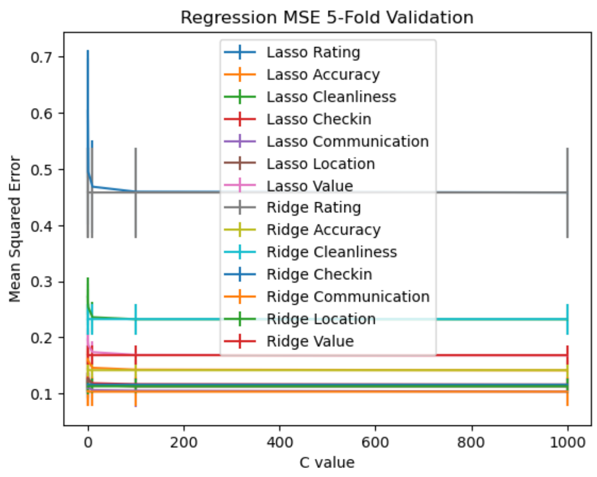


(a) 5 Folds cross validation Q selection on all scores

From the figure, we can see that the Mean Squared Error for each method and Score we are trying to evaluate only increased the error for Q = 1 to Q = 2. The ideal Q in all scenarios is

Q = 1. The error only increases or stays the same according to our graph. In these scenarios, choosing a smaller Q is best as the simplest model will generalize better. With so many features, choosing higher order polynomials will increase the training time dramatically as well.

Now that the ideal Q had been chosen, I ran 5 fold cross validation again on each score with Q = 1 to determine the ideal hyper parameter C. C is important to choose as it will affect how well the model will generalize to other data sets. Choosing a high C can lead to overfitting while choosing a low C can lead to underfitting, so we must use cross validation to determine the ideal C. Running 5 folds cross validation on Q = 1 for our scores and models gives the following results:



(a) 5 Folds cross validation C selection on all scores. Q = 1

Regression Cross Validation Mean Squared Error								
Penalty	Output Label	C=0.001	C=0.01	C=0.1	C=1	C=10	C=100	C=1000
Lasso	Rating	0.6032	0.6005	0.5911	0.4949	0.4684	0.4595	0.4581
Lasso	Accuracy	0.1619	0.1619	0.1588	0.1577	0.1458	0.1426	0.1415
Lasso	Cleanliness	0.273	0.273	0.2634	0.2537	0.2359	0.2324	0.2323
Lasso	Checkin	0.1278	0.1278	0.1252	0.1245	0.1184	0.1167	0.116
Lasso	Communication	0.1149	0.1149	0.1095	0.1112	0.1062	0.1053	0.1035
Lasso	Location	0.1215	0.1215	0.1207	0.1272	0.1155	0.1126	0.1129
Lasso	Value	0.1965	0.1965	0.1928	0.1863	0.1737	0.1689	0.1681
Ridge	Rating	0.4585	0.4575	0.4579	0.4581	0.4581	0.4581	0.4581
Ridge	Accuracy	0.1421	0.1416	0.1416	0.1416	0.1416	0.1416	0.1416
Ridge	Cleanliness	0.2324	0.2325	0.2326	0.2326	0.2326	0.2326	0.2326
Ridge	Checkin	0.1162	0.116	0.116	0.116	0.116	0.1159	0.1159
Ridge	Communication	0.1041	0.1034	0.1034	0.1034	0.1034	0.1034	0.1034
Ridge	Location	0.1127	0.1129	0.1129	0.1129	0.1129	0.1129	0.1129
Ridge	Value	0.1686	0.1683	0.1683	0.1684	0.1684	0.1684	0.1684

Based on the above table and plot, we can determine the ideal C value for each regression model type and score output. In all scenarios for Ridge regression, the Mean Squared Error does not change significantly for our wide choices, so we will choose a default of 1 to not favor overfitting or underfitting.

The lasso regression MSE are different, but they differ slightly. In all scenarios for the scores and their mean squared errors, the drop in error is lowest at 10, meaning from 10 to 100 has almost no reduction in mean squared error. Since the scale of the mean squared errors is so small, it is difficult to assess what "Significant" means in this context, but we will choose C = 10 for the ideal Lasso C hyper parameter for all lasso models.

The final step was to train our final model on the training data with our tuned hyper parameters and evaluate against the validation data. As a baseline, we will compare the mean squared error against a dummy model that predicts the average of the output. Below is the result.

Validation Results Mean Squared Error		
Methodology	Output Label	Mean Squared Error
Lasso	Rating	0.4057
Ridge	Rating	0.4052
Average (Dummy)	Rating	0.5259
Lasso	Accuracy	0.1573
Ridge	Accuracy	0.1565
Average (Dummy)	Accuracy	0.1685
Lasso	Cleanliness	0.297
Ridge	Cleanliness	0.2939
Average (Dummy)	Cleanliness	0.3517
Lasso	Checkin	0.1547
Ridge	Checkin	0.1493
Average (Dummy)	Checkin	0.1687
Lasso	Communication	0.1174
Ridge	Communication	0.1143
Average (Dummy)	Communication	0.136
Lasso	Location	0.1346
Ridge	Location	0.1289
Average (Dummy)	Location	0.1491
Lasso	Value	0.1446
Ridge	Value	0.1406
Average (Dummy)	Value	0.1834

The following weights were assigned to our models as the top 3 most positive and most negative weights for the Lasso and Ridge regression models that were trained.

Top 3 + and - Lasso Weights for AirBnb Scores	
Feature	Weight
Rating	_
years_since_last_review	-0.064725
accommodates	-0.000428
availability_60	-0.000237
number_of_reviews	0.002128
calculated_host_listings_count_private_rooms	0.005008
review_sentiment_average	0.080782
Accuracy	_
calculated_host_listings_count	-0.000947
calculated_host_listings_count_entire_homes	-0.000743
maximum_maximum_nights	-0.000204
description_sentiment	0.000243
number_of_reviews_ltm	0.000249
review_sentiment_average	0.027266
Cleanliness	_
years_since_last_review	-0.016978
calculated_host_listings_count_shared_rooms	-0.001174
minimum_minimum_nights	-0.000378
description_sentiment	0.000804
neighborhood_overview_sentiment	0.001987
review_sentiment_average	0.037428
Checkin	_
host_total_listings_count	-0.001113
maximum_maximum_nights	-0.000343
calculated_host_listings_count	-0.000202
maximum_nights_avg_ntm	0.000355
host_listings_count	0.001485
review_sentiment_average	0.018799
Communication	_
host_total_listings_count	-0.001627
calculated_host_listings_count	-0.000695
maximum_maximum_nights	-0.000203
maximum_nights_avg_ntm	0.000176
host_listings_count	0.002122
review_sentiment_average	0.019945
Location	_
years_since_last_review	-0.006655
host_about_sentiment	-0.000654
host_total_listings_count	-0.000471
description_sentiment	0.000904
calculated_host_listings_count_entire_homes	0.001418
review_sentiment_average	0.019731
Value	_
years_since_last_review	-0.003009
calculated_host_listings_count_entire_homes	-0.00203
calculated_host_listings_count	-0.000825
maximum_nights_avg_ntm	0.000115
number_of_reviews_ltm	0.00035
review_sentiment_average	0.035336

Top 3 + and - Lasso Weights for AirBnb Scores	
Feature	Weight
Rating	_
years_since_last_review	-0.093564
host_acceptance_rate	-0.059101
host_response_rate	-0.036529
has_availability	0.299279
host_phone_verified	0.356342
bathrooms	2.490918
Accuracy	_
bathrooms	-0.195457
host_acceptance_rate	-0.072593
has_availability	-0.060221
bedrooms	0.018351
review_sentiment_average	0.029687
host_is_superhost	0.036414
Cleanliness	_
host_phone_verified	-0.124327
host_acceptance_rate	-0.068584
bathrooms	-0.067051
host_email_verified	0.036532
review_sentiment_average	0.03965
host_is_superhost	0.052985
Checkin	_
bathrooms	-0.05681
host_acceptance_rate	-0.033792
host_has_profile_pic	-0.033017
bedrooms	0.01419
review_sentiment_average	0.021248
host_is_superhost	0.033448
Communication	_
bathrooms	-0.157551
host_acceptance_rate	-0.067296
host_has_profile_pic	-0.022312
review_sentiment_average	0.022221
host_is_superhost	0.032306
host_phone_verified	0.049322
Location	_
bathrooms	-0.219926
host_acceptance_rate	-0.055176
host_phone_verified	-0.04251
review_sentiment_average	0.022629
host_is_superhost	0.022808
has_availability	0.118336
Value	_
host_acceptance_rate	-0.079527
host_phone_verified	-0.065434
accommodates	-0.04141
review_sentiment_average	0.038093
host_is_superhost	0.042816
bedrooms	0.046099

Due to the large number of features, I will discuss the most important from the models. From the weights above in Ridge regression, it is clear that the average sentiment plays the largest role in almost all the scores for contributing to a positive score. A positive average score for the reviews tends to lead for a larger review score. Conversely, the years since the negative weight on the years since the last review will negatively affect the predicted score. This means that properties that have not been reviewed in a long time tend to have lower scores. The number of reviews, number of listings by the host, and sentiment description of the listing tend to impact the score positively.

For the lasso regression, features specifically about the host such as host_is_superhost and being verified are very important weights for scores regarding accuracy, checkin, cleanliness and location. Number of bathrooms is very important for the rating score, but has a negative weight associated with many of the other scores. Again with lasso regression, the sentiment average of all the reviews plays in important part in getting higher review scores in all the scores we are trying to predict.

In conclusion, the lasso and ridge regression models are slightly better than the dummy regression method. Due to the tight window and range of all reviews being on a scale of 1-5, it is uncertain as to how much better the regression methods are than the dummy regression, but there is measurably less error associated with them. This would suggest that it is possible to predict the air bnb score of a listing based on some data provided by air bnb.

(a) With logistic regression, we are attempting to find the best possible "line" that linearly separates our data in a binary classifier. For high dimensional data, this is a hyperplane. We calculate our weights based on some training data to predict whether an input is of class A or B and some threshold, for example.

Error can be easily introduced when the scale of some input feature values differ in orders of magnitude. The regression model during training may give the highest weight to those features making all other features negligible and thus resulting in incorrect predictions. An example of this is suppose a dataset has the timestamp of an email as a feature in classifying whether or not an email is spam. When an email is sent may be an indicator of whether or not an email is spam. If the timestamp is in unix time, those values are all very very large (1672430250 as I write this) which may skew the results drastically compared to another feature that has the char count of the email (which could be in terms of hundreds). Scaling these features would be required before hand to result in a better model to not give all the weight to such large features as unix time.

Another issue could arise in logistic regression from imbalanced data. In the same example as before with spam emails, most emails sent in the world are not spam. For arguments sake, suppose it is 90% of emails are real, while the rest are spam. The very nature of the data itself is skewed and there may not be enough training examples. The model may be trained so that it will just predict the most common class which will result in 10% error, which may appear to be good but is not actually better than a dummy baseline.

(b) A kNN classifier has the advantage of being a simple model. An MLP is far more complex than a KNN classifier and may not provide any significant improvements. In machine learning, the simplest model is often the best one.

A KNN classifier is also easy and fast to train with the cost function being convex. Depending on how deep the MLP is, it will take a much longer time to train which may not be desirable in certain scenarios.

Some disadvantages with KNN are that KNN does not perform well for areas we do not have data on, such as outliers. An MLP classifier will be able to generalize better beyond the initial data set.

A KNN classifier is effective for small data sets, but does not perform well for larger data sets. An MLP does not suffer from this. As the data size increases for a KNN classifier, it becomes more expensive to compute the distances over large data sets. An MLP will only need to be trained once and then can make predictions in constant time.

(c) K-folds cross validation allows us to evaluate how well our model predicts on new data. To do this, we split our data into training and test data randomly into K pieces and train our model on K-1 pieces and evaluate our model on that last piece. We do this for each "fold". It is up to the programmer to decide this "K" value. There is a trade off when choosing K. A higher K means we will have more data for a better model, but less testing which can introduce error and noise. If K is large, we also need to compute the model K times. A balance must be struck between the training and testing data, and k=5 or k=10 are evidently good values to use for validation.

(d) Lagged output values can be used as features for determining future predictions due to the nature of time series events. Things such as stock price or the number of available bicycles at a shared bike station are directly related to the previous data point. The count of bikes will not jump from 10 to 100 in a 1 minute window, but may jump from 10 to 12 in that same time period. This can be a useful feature for determining future predictions based on time series data.

Appendix

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import PolynomialFeatures
from matplotlib import cm
import math
import matplotlib.patches as mpatches
import sys
import copy
from sklearn.model_selection import KFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.dummy import DummyClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_curve
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from sklearn import preprocessing

 class ScatterData:
    def __init__(self,x,y,color,label,marker,alpha):
        self.x = x
        self.y = y
        self.color = color
        self.label = label
        self.marker = marker
        self.alpha = alpha
class PlotData:
    def __init__(self,x,y,color,label,alpha):
        self.x = x
        self.y = y
        self.color = color
        self.label = label
        self.alpha = alpha
class HistogramData:
    def __init__(self,data,numBins):
        self.data = data
        self.numBins = numBins

 def PlotHelper(title, xLabel, yLabel, scatterData = None, plotData = None, histogramData = None):
    fig = plt.figure()
    ax= plt.axes()
    ax.set_xlabel(xLabel)
    ax.set_ylabel(yLabel)
    ax.set_title(title, loc='left')

    #(self,x,y,color,label,marker,alpha):
    if scatterData is not None:
        for data in scatterData:
            ax.scatter(data.x,data.y, label = data.label, color=data.color, alpha = data.alpha, marker =data.

    if plotData is not None:
        for data in plotData:
            ax.plot(data.x,data.y, label = data.label, color=data.color, alpha = data.alpha)

    if histogramData is not None:
        plt.hist(histogramData.data, bins=histogramData.numBins)

    plt.legend()
    plt.show()

 def MeanSquareErrorPlot(title, xLabel, xVals, mse, std):
    fig = plt.figure()
    ax = plt.axes()
    ax.set_title(title)
    plt.xlabel(xLabel)
    plt.ylabel("Mean Squared Error")
    plt.errorbar(xVals, mse, yerr=std)
    plt.show()

def MeanSquareErrorPlotAlt(title, xLabel, xVals, mseList, stdList, labels):
    fig = plt.figure()
    ax = plt.axes()
    ax.set_title(title)
    plt.xlabel(xLabel)
    plt.ylabel("Mean Squared Error")
    for i,mse in enumerate(mseList):
#          print(labels[i], xValsList, mseList[i],stdList[i])
        plt.errorbar(xVals, mseList[i], label=labels[i], yerr=stdList[i])

    plt.legend()
    plt.show()

class Dataset:
    def __init__(self, file, splitPercentage = .9, Debug = False):
        df = pd.read_csv(file)
        if Debug:
            print(df.head())
            print(df.info())

        numRowsTrain = int(splitPercentage * df.shape[0])
        numRowsValidate = df.shape[0] - numRowsTrain
        #https://stackoverflow.com/questions/29576430/shuffle-dataframe-rows
        df = shuffle(df, random_state = 0)
        df.reset_index(inplace=True, drop=True)
        self.X = df.iloc[:,1:]
        self.y = df.iloc[:,0]
        self.ColumnNames = df.columns[1:]


        self.trainX = self.X.iloc[:numRowsTrain]
        self.validateX = self.X.iloc[numRowsTrain:]
        self.trainY = self.y.iloc[:numRowsTrain]
        self.validateY = self.y.iloc[numRowsTrain:]

        self.xPolys = {}
        self.trainxPolys= {}
        self.validatexPolys = {}
        self.polynomialFeatureNames = {}
    def AddPolynomialFeatures(self, degree):
        pf = PolynomialFeatures(degree)
        self.xPolys[degree] = pf.fit_transform(self.X)
        self.trainxPolys[degree] = pf.fit_transform(self.trainX)
        self.validatexPolys[degree] = pf.fit_transform(self.validateX)
        self.polynomialFeatureNames[degree] = pf.get_feature_names_out(self.ColumnNames)

    def PrintColumns(self):
        for name in self.ColumnNames:
            print(name)

class MLModel:
    def __init__(self):
        self.thetas = []
        self.type = None
        self.yPred = None
        self.model = None
    def TrainModel(self, ModelType, x, y, c = None, K = None):
        assert(self.type == None and ModelType in ["Lasso", "Ridge", "KNN"])
        self.type = ModelType
        if ModelType == "Lasso":
            self.model = linear_model.Lasso(alpha=(1/(2 * c)))
        elif ModelType == "Ridge":
            self.model = linear_model.Ridge(alpha=(1/(2 * c)))
        elif ModelType == "KNN":
            self.model = KNeighborsRegressor(n_neighbors = K)
            #assert (False)
        print("Fitting " + self.type)
        self.model.fit(x, y)

        if ModelType in ["Lasso", "Ridge"]:
            self.thetas.append(self.model.intercept_)
            for data in self.model.coef_:
                self.thetas.append(data)

    def KFoldsValidation(self, ModelType, x, y, hyperparameter = None, folds = 5):
        kf = KFold(n_splits = folds)
        assert(self.type == None and ModelType in ["Lasso", "Ridge", "KNN"])
        self.type = ModelType
        self.meanError = []
        self.stdError = []
        # Use current polynomial features and
        # C value to perform k folds validation
        if ModelType == "Lasso":
            self.model = linear_model.Lasso(alpha=(1/(2 * hyperparameter)))
        elif ModelType == "Ridge":
            self.model = linear_model.Ridge(alpha=(1/(2 * hyperparameter)))
        elif ModelType == "KNN":
            return self.KFoldsKNN(x,y,hyperparameter, folds)


        temp = []
        for train,test in kf.split(x):
            self.model.fit(x[train], y[train])
            yPred = self.model.predict(x[test])
            # append the F1 Score for the currently trained model
            temp.append(mean_squared_error(y[test],yPred))

        self.meanError = np.array(temp).mean()
        self.stdError = np.array(temp).std()
        return self.meanError, self.stdError

    def KFoldsKNN(self, x, y, K, folds = 5):
        kf = KFold(n_splits = folds)
        self.model = KNeighborsRegressor(n_neighbors = K)
        self.meanError = []
        self.stdError = []

        temp = []
        for train,test in kf.split(x):
            self.model.fit(x[train], y[train])
            yPred = self.model.predict(x[test])

            temp.append(mean_squared_error(y[test],yPred))

        self.meanError = np.array(temp).mean()
        self.stdError = np.array(temp).std()
        return self.meanError, self.stdError
    def Predict(self, x, y):
        assert self.type != None
        yPred = self.model.predict(x)
        mse = mean_squared_error(y,yPred)
        return mse

    def PrintWeights(self, names):
        assert(self.type in ["Lasso", "Ridge"])
        weights = self.thetas[1:]
        print(len(names), len(weights))
        for i in range(len(weights)):
            print(names[i], weights[i])

review_scores_rating_data = Dataset("./scrubbed-data-rating.csv")
review_scores_accuracy_data = Dataset("./scrubbed-data-accuracy.csv")
review_scores_cleanliness_data = Dataset("./scrubbed-data-cleanliness.csv")
review_scores_checkin_data = Dataset("./scrubbed-data-checkin.csv")
review_scores_communication_data = Dataset("./scrubbed-data-communication.csv")
review_scores_location_data = Dataset("./scrubbed-data-location.csv")
review_scores_value_data = Dataset("./scrubbed-data-value.csv")

#create our datasets with their label
datasets = {
    "Rating" : review_scores_rating_data,
    "Accuracy" : review_scores_accuracy_data,
    "Cleanliness" : review_scores_cleanliness_data,
    "Checkin" : review_scores_checkin_data,
    "Communication" : review_scores_communication_data,
    "Location" : review_scores_location_data,
    "Value" : review_scores_value_data
}
datasets

 # Qs = [1,2,3,4,5]
Qs = [1,2]
for datasetKey in datasets:
    dataset = datasets[datasetKey]
    for q in Qs:
        dataset.AddPolynomialFeatures(q)
    dataset.trainY.mean()
    dataset.trainX

for datasetKey in datasets:
    dataset = datasets[datasetKey]
    print(dataset)

CVals = [0.01,0.1,1,10,100,1000]
ModelTypes = ["Ridge", "Lasso"]

Models = []
defaultC = 1
MSEList = []
STDList = []
LabelList = []
# choose q then choose c
for mType in ModelTypes:
    for datasetKey in datasets:
        dataset = datasets[datasetKey]
        mse = []
        std = []
        hyperparameterLabel = ""
        hyperparameterList = []
        if mType == "KNN":
            hyperparameterLabel = "K"
            hyperparameterList = KVals
            for k in KVals:
                print(datasetKey,"K =",k)
                m = MLModel()
                x = dataset.trainX
                y = dataset.trainY
                currMSE, currSTD = m.KFoldsValidation(mType,x,y,k)
                mse.append(currMSE)
                std.append(currSTD)
                Models.append(m)
        else:
            hyperparameterLabel = "Q"
            hyperparameterList = Qs
            for q in dataset.trainxPolys:
                print(datasetKey, mType, "Q =", q)
                m = MLModel()
                x = dataset.trainxPolys[q]
                y = dataset.trainY
                currMSE, currSTD = m.KFoldsValidation(mType,x,y,defaultC)
                mse.append(currMSE)
                std.append(currSTD)
                Models.append(m)
        MSEList.append(mse)
        STDList.append(std)
        LabelList.append(mType + " " + datasetKey)

#         MeanSquareErrorPlot(mType + " regression. 5 FoldsCV for " + hyperparameterLabel + ". " + str(dataset

MeanSquareErrorPlotAlt("Regression MSE 5-Fold Validation", "Q value", Qs, MSEList, STDList, LabelList)

idealQ = 1
MSEListC = []
STDListC = []
LabelListC = []

ModelsCV = []
ModelTypes = ["Lasso", "Ridge"]
Cs = [0.001,0.01,0.1,1,10,100, 1000]
for mType in ModelTypes:
    for datasetKey in datasets:
        dataset = datasets[datasetKey]
        mse = []
        std = []
        hyperparameterLabel = "C"
        hyperparameterList = Cs
        for c in Cs:
            print(mType, "C =", c)
            m = MLModel()
            x = dataset.trainxPolys[idealQ]
            y = dataset.trainY
            currMSE, currSTD = m.KFoldsValidation(mType,x,y,c)
            mse.append(currMSE)
            std.append(currSTD)
            ModelsCV.append(m)
        MSEListC.append(mse)
        STDListC.append(std)
        LabelListC.append(mType + " " + datasetKey )

MeanSquareErrorPlotAlt("Regression MSE 5-Fold Validation", "C value", Cs, MSEListC, STDListC, LabelListC)

print(Cs)
for i,x in enumerate(MSEListC):
    cs = ""
    for element in x:
        cs += str(round(element,4)) + " & "
    cs = cs.strip(" & ")

    print(LabelListC[i].split(" ")[0],"&",LabelListC[i].split(" ")[1], "&", cs,"\\cr")
```

```
idealCLasso = 10
idealCRidge = 1
idealQ = 1
LassoFinalModels= {}
RidgeFinalModels = {}
from sklearn.dummy import DummyRegressor
from sklearn.metrics import mean_squared_error
for datasetKey in datasets:
    dataset = datasets[datasetKey]
for datasetKey in datasets:
    dataset = datasets[datasetKey]
    x = dataset.trainxPolys[idealQ]
    y = dataset.trainY

    Lasso = MLModel()
    Lasso.TrainModel("Lasso", x,y, c = idealCLasso)
    LassoFinalModels[datasetKey] = Lasso
    print("Lasso & " + datasetKey , "&", str(round(Lasso.Predict(dataset.validatexPolys[idealQ], dataset.valid


    Ridge = MLModel()
    Ridge.TrainModel("Ridge", x,y, c = idealCRidge)
    RidgeFinalModels[datasetKey] = Ridge
    print( "Ridge & " + datasetKey , "&", str(round(Ridge.Predict(dataset.validatexPolys[idealQ], dataset.vali


    dummy = DummyRegressor(strategy="mean").fit(dataset.trainX, dataset.trainY)
    dummyPred = dummy.predict(dataset.validateX)
    print("Average (Dummy) & "+ datasetKey, "&", str(round(mean_squared_error(dataset.validateY,dummyPred),4))
    print("\\\\ hline")

for key in LassoFinalModels:
    model = LassoFinalModels[key]
#      datasets[key].ColumnNames
    print(key)
    dataset = datasets[key]
#      names.insert(0,"1")
#      print(len(names),names)
#      model.PrintWeights(names)
    pairs = list(zip(model.thetas[1:], dataset.polynomialFeatureNames[idealQ]))
    pairs.sort(key=lambda a: a[0])
    for pair in pairs:
        print(str(round(pair[0],6)), pair[1])
    print("____")

for key in RidgeFinalModels:
    model = RidgeFinalModels[key]
#      datasets[key].ColumnNames
    print(key,"& _ \\cr")
    dataset = datasets[key]
#      names.insert(0,"1")
#      print(len(names),names)
#      model.PrintWeights(names)
    pairs = list(zip(model.thetas[1:], dataset.polynomialFeatureNames[idealQ]))
    pairs.sort(key=lambda a: a[0])

    length = len(pairs)
    cutoff = 3
    for i,pair in enumerate(pairs):
        if i < 3 or length - i <= 3:
            print(pair[1], "&",str(round(pair[0],6)), "\\cr")
    print("\\hline")
```