**Homework 2 Python**

# **DATA 624-01 Group 3**

# **Z. Palmore, K. Popkin, K. Potter, C. Nan, J. Ramalingam**

# 

All of the responses in this document are aimed at providing context to the differences in processes and outcomes using python instead of R. This was done to avoid duplication of content between this document and the homework 2 submitted in R. Use the link below to find the Jupyter Notebook that contains all the code in this code with additions.

[LINK TO CODE](https://github.com/KevinJpotter/msds/upload/main/624/Homework)

Key:

Homework problems are stated in blue text

Answers are in black text

KJ 6.3

A chemical manufacturing process for a pharmaceutical product was discussed in Sect. 1.4. In this problem, the objective is to understand the relationship between biological measurements of the raw materials (predictors),measurements of the manufacturing process (predictors), and the response of product yield. Biological predictors cannot be changed but can be used to assess the quality of the raw material before processing. On the other hand, manufacturing process predictors can be changed in the manufacturing process. Improving product yield by 1% will boost revenue by approximately one hundred thousand dollars per batch:

Question

(a) Start R and use these commands to load the data:

Code

|  |
| --- |
| import pandas as pd df = pd.read\_csv("https://raw.githubusercontent.com/palmorezm/msds/main/624/Data/hw\_2\_data/ChemicalManufacturingProcess.csv",  index\_col=0) df.head() |

Response

Python’s main package for reading and manipulating data is pandas.

Question

(b) A small percentage of cells in the predictor set contain missing values. Use an imputation function to fill in these missing values (e.g., see Sect. 3.8).

Code

|  |
| --- |
| df = df.fillna(df.median()) |

Response

We filled the missing data in our R assignment with the median, we can use pandas ‘fillna’ method to do this in one line of code.

Question

(c) Split the data into a training and a test set, pre-process the data, and tune a model of your choice from this chapter.

Code

|  |
| --- |
| import statsmodels.api as sm from sklearn.metrics import r2\_score, mean\_squared\_error corelated\_columns = list(df.corr()['Yield'].abs().sort\_values(ascending=False).head(8).index) df = df[corelated\_columns] bound = int(len(df) \*.7) X\_train, X\_test, y\_train, y\_test = train\_test\_split(df.drop(columns=['Yield']), df['Yield'], test\_size=0.25, random\_state=42) X\_train = sm.add\_constant(X\_train) X\_test = sm.add\_constant(X\_test) lr = sm.OLS(y\_train, X\_train) lr = lr.fit() |

Response

We use the statsmodel OLS package to create our multivariate linear regression model. The summary of this model will look very similar to that of it’s R counterpart. We split the data using a package from the sklearn package. This is a very common package used to do ML in python.

Question

(d) Predict the response for the test set.What is the value of the performance metric and how does this compare with the resampled performance metric on the training set?

Code

|  |
| --- |
| from sklearn.metrics import r2\_score from sklearn.model\_selection import cross\_val\_score  from sklearn.base import BaseEstimator, RegressorMixin class SMWrapper(BaseEstimator, RegressorMixin):  """ A universal sklearn-style wrapper for statsmodels regressors """  def \_\_init\_\_(self, model\_class, fit\_intercept=True):  self.model\_class = model\_class  self.fit\_intercept = fit\_intercept  def fit(self, X, y):  if self.fit\_intercept:  X = sm.add\_constant(X)  self.model\_ = self.model\_class(y, X)  self.results\_ = self.model\_.fit()  def predict(self, X):  if self.fit\_intercept:  X = sm.add\_constant(X)  return self.results\_.predict(X) y\_test\_preds = lr.predict(X\_test) print("R2 for the model is: ",r2\_score(y\_train, y\_train\_preds)) print("Resampled R2 is: ", cross\_val\_score(SMWrapper(sm.OLS), X\_test, y\_test, cv=5)) |

R2 for the model is: 0.626950822599345

Resampled R2 is: [ 0.59808165 0.71856995 -0.36147153 0.51278635 -0.13033599]

Response

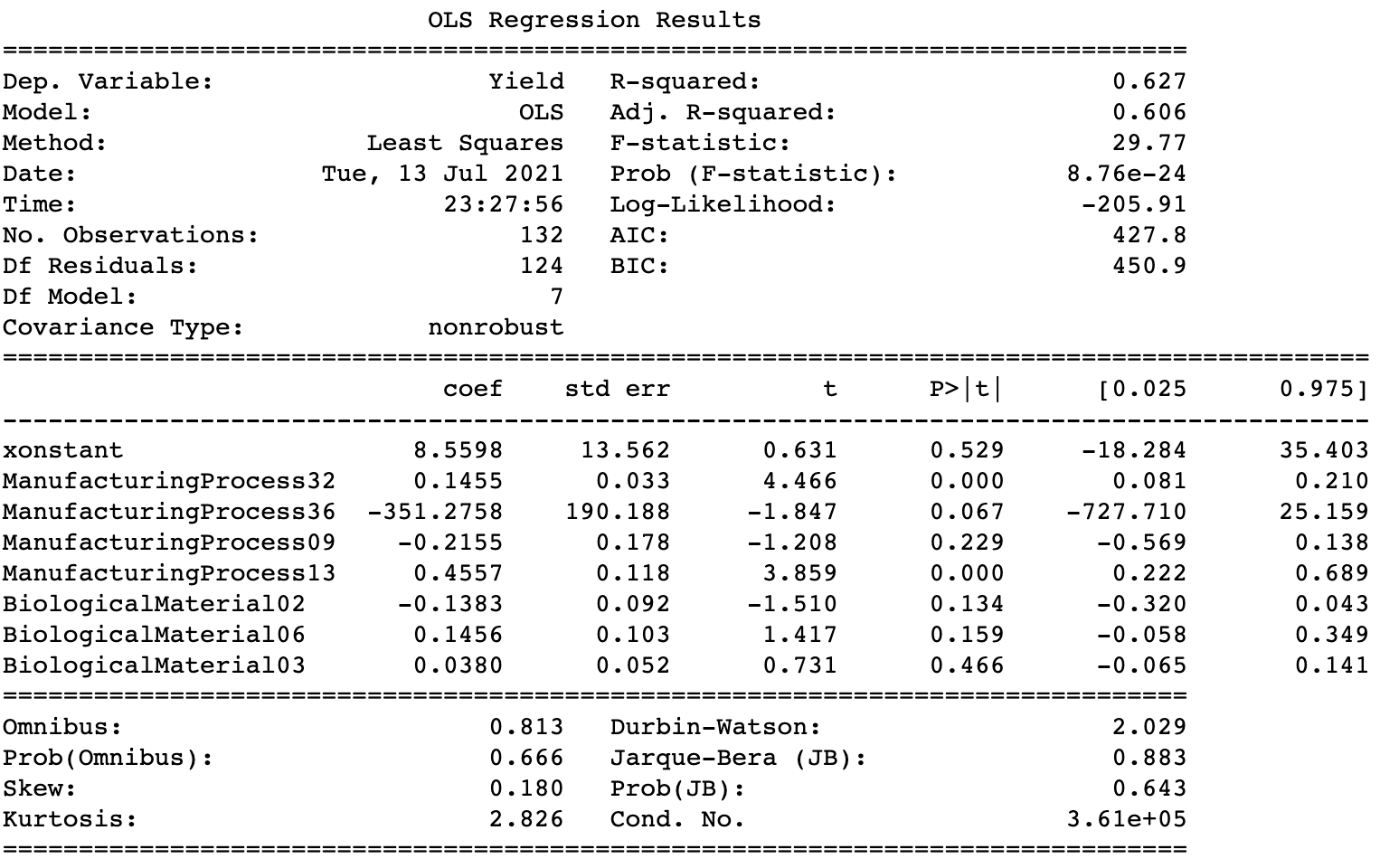
Using this process our model performance is almost identical to what we saw from the R model, slightly better. This .01 increase in performance would probably be due to the difference in sampling. The sklearn package creates a random sample which is good for avoiding bias from data entry. We also evaluate the cross validated scores to see the performance on resampled data. The varied results here show how the sample can have a big impact on performance.

Question

(e) Which predictors are most important in the model you have trained? Do either the biological or process predictors dominate the list?

Code

|  |
| --- |
| print(lr.summary(xname=["xonstant", "ManufacturingProcess32", "ManufacturingProcess36", "ManufacturingProcess09", "ManufacturingProcess13", "BiologicalMaterial02", "BiologicalMaterial06", "BiologicalMaterial03"])) |



Response

In our model we only used the 7 highest correlated features as we did in the R example. In this case the dominant feature is the Manufacturing Process 36 by a wide margin. Seeing how this outweighs all of the other features by such a wide margin, this would not be a good model to use in production. You would want to see evenly distributed weights to avoid potential drift and bias down the road.

Question

(f) Explore the relationships between each of the top predictors and the response.

How could this information be helpful in improving yield in future runs of the manufacturing process?

Code

|  |
| --- |
| display(df.corr()['Yield'].abs().sort\_values(ascending=False).head(8)) |

ManufacturingProcess32 0.608332

ManufacturingProcess36 0.525095

ManufacturingProcess13 0.503680

ManufacturingProcess09 0.503471

BiologicalMaterial02 0.481516

BiologicalMaterial06 0.478163

BiologicalMaterial03 0.445086

Response

Similar to our R example we use the correlations to see the relationship between a variable and target variable. It’s important to look at covariance as well when creating models as adding features with high covariance can lead to a biased model. The other interesting number to look at is the high p-value found on the regression output for many variables. This leads us to believe they are not good predictors.

KJ 7.2

Friedman (1991) introduced several benchmark data sets create by simulation. One of these simulations used the following nonlinear equation to create data: y = 10 sin(πx1x2) + 20(x3 − 0.5)2 + 10x4 + 5x5 + N(0, σ2) where the x values are random variables uniformly distributed between [0, 1] (there are also 5 other non-informative variables also created in the simulation).The package mlbench contains a function called mlbench.friedman1 that simulates these data:

Question

Tune several models on these data.

Code

|  |
| --- |
| import pandas as pd X\_train = pd.read\_csv('https://raw.githubusercontent.com/palmorezm/msds/main/624/Data/hw\_2\_data/trainingData.csv', index\_col=0) y\_train = pd.read\_csv('https://raw.githubusercontent.com/KevinJpotter/msds/main/624/Data/hw\_2\_data/trainingData\_y.csv', index\_col=0) X\_test = pd.read\_csv('https://raw.githubusercontent.com/palmorezm/msds/main/624/Data/hw\_2\_data/testData.csv', index\_col=0) y\_test = pd.read\_csv('https://raw.githubusercontent.com/KevinJpotter/msds/main/624/Data/hw\_2\_data/testData\_y.csv', index\_col=0) display(X\_train.head()) display(y\_train.head())  from sklearn.neighbors import KNeighborsRegressor from sklearn.model\_selection import cross\_val\_score knn = KNeighborsRegressor(n\_neighbors=19) knn.fit(X\_train, y\_train) y\_preds = knn.predict(X\_test) print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RESAMPLED TEST DATA KNN\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*") print(cross\_val\_score(knn, X\_test, y\_test, cv=5))  from sklearn.svm import SVR svr = SVR().fit(X\_train, y\_train.values.ravel()) y\_preds = svr.predict(X\_test) print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RESAMPLED TEST DATA SVM\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*") print(cross\_val\_score(svr, X\_test, y\_test.values.ravel(), cv=5))  from keras.models import Sequential from keras.layers import Dense model = Sequential() model.add(Dense(12, input\_dim=10, activation='relu')) model.add(Dense(8, activation='relu')) model.add(Dense(1, activation='linear')) model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy']) model.fit(X\_train, y\_train.values.ravel(), epochs=500, batch\_size=10, verbose=0) y\_preds = model.predict(X\_test) print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*TEST DATA NNET\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*") print(r2\_score(y\_test, y\_preds)) |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RESAMPLED TEST DATA KNN\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[0.78774556 0.77202093 0.78470723 0.78783084 0.7749057 ]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RESAMPLED TEST DATA SVM\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[0.9192057 0.89285931 0.90965653 0.91206067 0.90412034]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*TEST DATA NNET\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-6.2295953175144465

Response

As our non linear options we use a neigherst neighbors, support vector, and small neural network. Python has a package called pyearth that uses the MARS algorithm but my version of python was not supported by the package and I was not able to use it for this problem. The results are all in line with what we saw in R aside from the neural network which in this case is poorly trained. Neural Networks often don’t perform well on small datasets.

Question

Which models appear to give the best performance?

Response

The model that gives the best performance is the Support Vector Regression averaging about .91 of explained variance, which is very high.

Question

Does MARS select the informative predictors (those named X1–X5)?

Response

I was not able to use the MARS equivalent in python due to dependency issues.

KJ 7.5

Exercise 6.3 describes data for a chemical manufacturing process. Use

the same data imputation, data splitting, and pre-processing steps as before

and train several nonlinear regression models.

MARS and KNN models were trained and tested with the chemical data.

Question

(a) Which nonlinear regression model gives the optimal resampling and test

set performance?

Code

|  |
| --- |
| import pandas as pd from sklearn.model\_selection import train\_test\_split df = pd.read\_csv("https://raw.githubusercontent.com/palmorezm/msds/main/624/Data/hw\_2\_data/ChemicalManufacturingProcess.csv", index\_col=0)  import statsmodels.api as sm from sklearn.metrics import r2\_score, mean\_squared\_error corelated\_columns = list(df.corr()['Yield'].abs().sort\_values(ascending=False).head(8).index) df = df.fillna(df.median()) X\_train, X\_test, y\_train, y\_test = train\_test\_split(df.drop(columns=['Yield']), df['Yield'], test\_size=0.25, random\_state=42)  from sklearn.neighbors import KNeighborsRegressor from sklearn.model\_selection import cross\_val\_score knn = KNeighborsRegressor(n\_neighbors=4) knn.fit(X\_train, y\_train) y\_preds = knn.predict(X\_test) print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RESAMPLED TEST DATA KNN\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*") print(cross\_val\_score(knn, X\_test, y\_test, cv=5))  from sklearn.svm import SVR svr = SVR().fit(X\_train, y\_train.values.ravel()) y\_preds = svr.predict(X\_test) print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RESAMPLED TEST DATA SVM\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*") print(cross\_val\_score(svr, X\_test, y\_test.values.ravel(), cv=5))  from sklearn.ensemble import RandomForestRegressor rf = RandomForestRegressor() rf.fit(X\_train, y\_train) y\_preds = rf.predict(X\_test) print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RESAMPLED TEST DATA RF\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*") print(cross\_val\_score(rf, X\_test, y\_test.values.ravel(), cv=5)) |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RESAMPLED TEST DATA KNN\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[ 0.08630118 -0.90936953 -0.6886088 0.34027675 -0.21919276]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RESAMPLED TEST DATA SVM\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[ 0.0173789 -0.48396686 0.12231512 0.10694668 -0.14187132]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RESAMPLED TEST DATA RF\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[0.36840078 0.5773526 0.321676 0.36329476 0.19210864]

Response

In this example we use KNN, SVM, and a tree-based model Random Forest. The tree based regressor gives us the best performance across the resampled data. Again we use the sklearn package to build the models and measure performance.

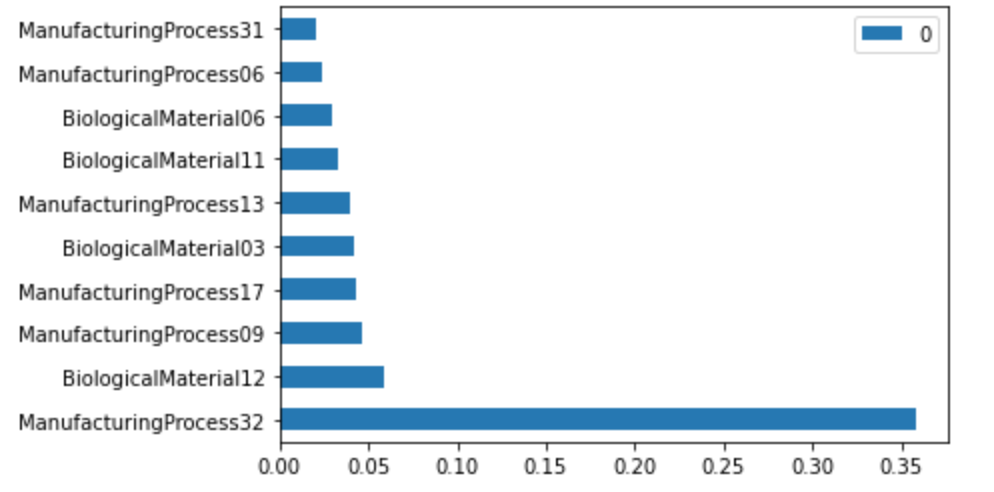
Question

(b) Which predictors are most important in the optimal nonlinear regression

model?

Code

|  |
| --- |
| pd.DataFrame(rf.feature\_importances\_,index= X\_train.columns).sort\_values(ascending=False, by=0).head(7).plot(kind='barh'); |



Response

The results of the most important features arever similar to what we saw from the MARS model in our R example. Process 32 is the number one by a large margin; this was also the case in the R example. The remaining 9 are mixed results compared to what we saw in R.

Question

Do either the biological or process variables dominate the list?

Response

Yes, the process variable dominated the importance with biological coming in a distant second.

Question

How do the top ten important predictors compare to the top ten predictors from the optimal linear model?

Response

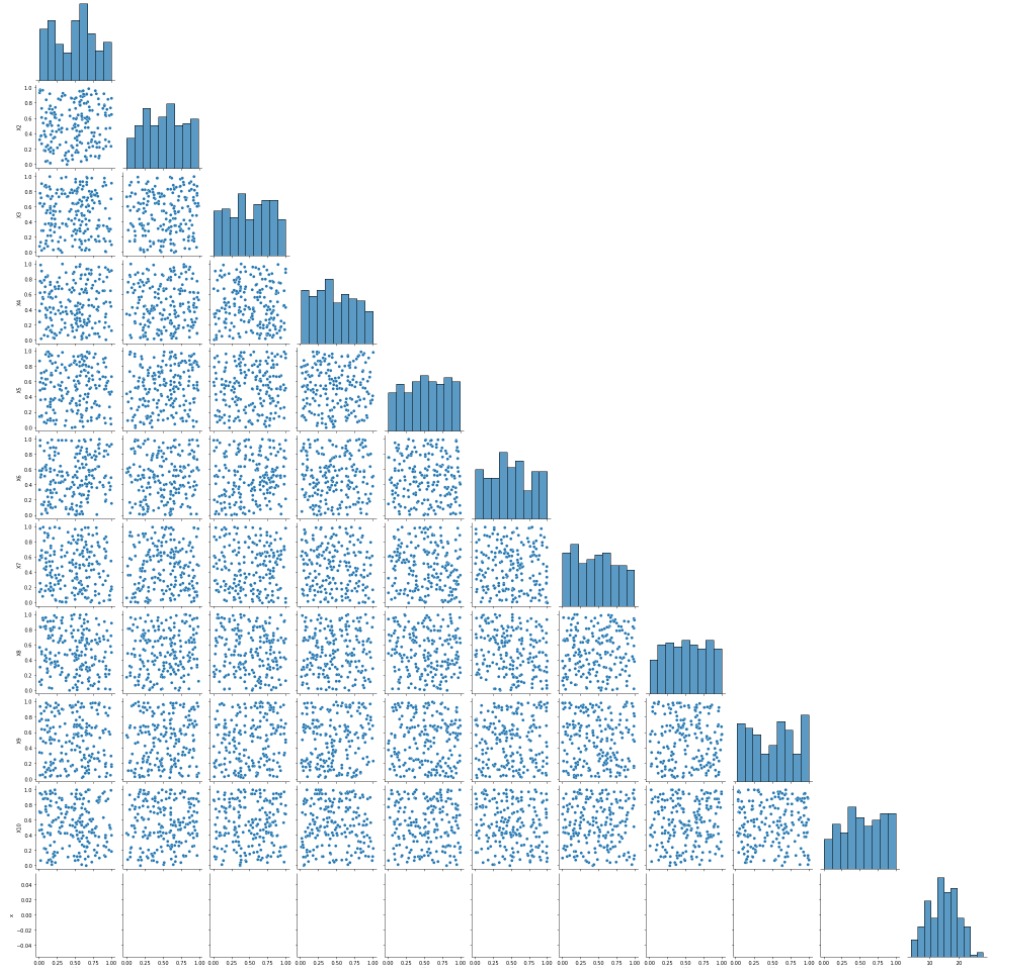
Like the R example we see many of the same variables having the biggest impact on the predictions.

Question

(c) Explore the relationships between the top predictors and the response for the predictors that are unique to the optimal nonlinear regression model. Do these plots reveal intuition about the biological or process predictors and their relationship with yield?

Code

|  |
| --- |
| import seaborn as sns sns.pairplot(pd.concat([X\_train, y\_train]), corner=True) |



Response

We use the pariplot method, from the seaborn package for data visualizations in python, to easily see the relationships between each variable. The dispersion in each plot shows there are not many linear relationships between variables or any trend to identify at all. This makes it difficult to model, especially using linear techniques.

# 

# 

# 

# KJ 8.1

8.1. Recreate the simulated data from Exercise 7.2:

Question

(a) Fit a random forest model to all of the predictors, then estimate the variable importance scores: Did the random forest model significantly use the uninformative predictors (V6 - V10)?

Code

|  |
| --- |
| import pandas as pd import numpy as np from sklearn.ensemble import RandomForestRegressor from sklearn.ensemble import GradientBoostingRegressor from sklearn import tree import sys  def build\_model(df\_train,model):  train\_x=[]  df\_importances = pd.DataFrame(columns=["Feature","Importance"])  for i in range(0,200):  if(len(df\_train.columns)==11): train\_x.append(np.array([df\_train.loc[i,"V1"],df\_train.loc[i,"V2"],df\_train.loc[i,"V3"],df\_train.loc[i,"V4"],df\_train.loc[i,"V5"],df\_train.loc[i,"V6"],df\_train.loc[i,"V7"],df\_train.loc[i,"V8"],df\_train.loc[i,"V9"],df\_train.loc[i,"V10"]]))  df\_importances.Feature=["V1","V2","V3","V4","V5","V6","V7","V8","V9","V10"]  else: train\_x.append(np.array([df\_train.loc[i,"V1"],df\_train.loc[i,"V2"],df\_train.loc[i,"V3"],df\_train.loc[i,"V4"],df\_train.loc[i,"V5"],df\_train.loc[i,"V6"],df\_train.loc[i,"V7"],df\_train.loc[i,"V8"],df\_train.loc[i,"V9"],df\_train.loc[i,"V10"],df\_train.loc[i,"V11"]])) df\_importances.Feature=["V1","V2","V3","V4","V5","V6","V7","V8","V9","V10","V11"]  train\_y = df\_train["y"]  if(model=="RF"):  rf = RandomForestRegressor(n\_estimators = 200, random\_state = 0)  rf.fit(train\_x, train\_y)  imp\_out = rf.feature\_importances\_  elif(model=="GBM"):  rf = tree.DecisionTreeRegressor()  rf.fit(train\_x, train\_y)  imp\_out = rf.feature\_importances\_  elif(model=="DT"):  model = GradientBoostingRegressor(max\_depth=1)  model.fit(train\_x,train\_y)  imp\_out=model.feature\_importances\_  else:  print("Not a Valid Model")  sys.exit()   imp\_list=[]  for i in range(0,len(imp\_out)):  imp\_list.append(imp\_out[i])  df\_importances.Importance=imp\_list  return(df\_importances) |

Response

The importance given to the predictors (V6-V10) is less and hence the usage is insignificant. We see the same results in our R model as well. Here we build a function that allows us to take in the data and easily compare the outputs of each model.

Question

(b) Now add an additional predictor that is highly correlated with one of the informative predictors. For example:

Fit another random forest model to these data. Did the importance score for V1 change? What happens when you add another predictor that is also highly correlated with V1?

Code

|  |
| --- |
| df\_train = pd.read\_csv("simulated.csv") importance\_p1 = build\_model(df\_train,"RF") importance\_p1  random\_variable=list(np.random.normal(2, 4, 200)\*0.1) new\_col = [] old\_col = df\_train["V1"] for x in range(0,200):  new\_col.append(old\_col[x]+random\_variable[x])  df\_train["V11"] = new\_col  random\_variable=list(np.random.normal(2, 4, 200)\*0.1) new\_col = [] old\_col = df\_train["V1"] for x in range(0,200):  new\_col.append(old\_col[x]+random\_variable[x]) df\_train["V11"] = new\_col importance\_p2 = build\_model(df\_train,"RF") importance\_p2 |

Feature Importance

0 V1 0.286612

1 V2 0.222170

2 V3 0.039425

3 V4 0.259745

4 V5 0.082464

5 V6 0.020679

6 V7 0.020911

7 V8 0.013882

8 V9 0.013041

9 V10 0.020054

10 V11 0.021018

Response

The Importance is split between predictors, the importance of V1 predictor is split across the correlated variable exactly the same to what we saw using R packages.

Question

(c) Use the cforest function in the party package to fit a random forest model using conditional inference trees. The party package function varimp can calculate predictor importance. The conditional argument of that function toggles between the traditional importance measure and the modified version described in Strobl et al. (2007). Do these importances show the same pattern as the traditional random forest model?

Response

Python doesn't have CFOREST implementation

Question

(d) Repeat this process with different tree models, such as boosted trees and Cubist. Does the same pattern occur?

Code

|  |
| --- |
| importance\_gbm = build\_model(df\_train,"GBM") importance\_gbm   importance\_dt = build\_model(df\_train,"DT") importance\_dt |

GBM Output :

Feature Importance

0 V1 0.287547

1 V2 0.282870

2 V3 0.046633

3 V4 0.233616

4 V5 0.067086

5 V6 0.009949

6 V7 0.015848

7 V8 0.004654

8 V9 0.005875

9 V10 0.017593

10 V11 0.028329

Decision Tree Output:

Feature Importance

0 V1 0.293014

1 V2 0.245595

2 V3 0.061552

3 V4 0.308648

4 V5 0.091190

5 V6 0.000000

6 V7 0.000000

7 V8 0.000000

8 V9 0.000000

9 V10 0.000000

10 V11 0.000000

Response

After running the two variants through our function we can see that the top rated predictor is V1 in GBM and V4 in Decision Tree. It’s also evident that V6 - V10 scores are very low in GBM and 0 in Decision Tree where the V11 score is 0 in Decision tree but higher in GBM. The pattern is different across the three models.

# KJ 8.2

Question

Use a simulation to show tree bias with different granularities.The simulation is created with two variables

1. X1 -> Lesser variance (Sequence of values)
2. X2 -> Higher Variance (Gaussian Random value)
3. Target Variable Y -> X1 + X2

Code

|  |
| --- |
| import random import numpy as np import pandas as pd from sklearn import tree from sklearn.model\_selection import train\_test\_split from sklearn.datasets import make\_classification from sklearn.tree import DecisionTreeClassifier,DecisionTreeRegressor from matplotlib import pyplot  random.seed(755) x = np.array([1,2]) X\_low=list(np.repeat(x, 300, axis=0)) X\_low X\_high=list(np.random.normal(2, 4, 600)) X\_high X = list() Y = list() for x in range(0,600):  Y.append(X\_low[x] + X\_high[x])  X.append(np.array([X\_low[x],X\_high[x]]))  clf = tree.DecisionTreeRegressor() clf = clf.fit(X, Y) # get importance importance = clf.feature\_importances\_ # summarize feature importance for i,v in enumerate(importance):  print('Feature: %0d, Score: %.5f' % (i,v)) # plot feature importance pyplot.bar([x for x in range(len(importance))], importance) pyplot.show() |

Feature: 0, Score: 0.01023

Feature: 1, Score: 0.98977

Response

The difference between two variables is significant and it shows the tree bias between two variables. Feature one has a much greater score and it’s easy to see the imbalance.

# KJ 8.3

8.3. In stochastic gradient boosting the bagging fraction and learning rate will govern the construction of the trees as they are guided by the gradient. Although the optimal values of these parameters should be obtained through the tuning process, it is helpful to understand how the magnitudes of these parameters affect magnitudes of variable importance. Figure 8.24 provides the variable importance plots for boosting using two extreme values for the bagging fraction (0.1 and 0.9) and the learning rate (0.1 and 0.9) for the solubility data. The left-hand plot has both parameters set to 0.1, and the right-hand plot has both set to 0.9:

Question

(a) Why does the model on the right focus its importance on just the first few of predictors, whereas the model on the left spreads importance across more predictors?

Response

There could be two reasons for the model on the right spreads importance on the first few predictors and the model on the left adds importance to few more predictors.

1. Bragging Fraction : It represents the data usage in each iteration of trees. The left hand plot has a bragging fraction of 0.1 which is low and only 10% of data is used for random sampling. Whereas , the right hand plot has a bragging fraction of 0.9 which is large and 90% of data is used on each iteration which is almost the full dataset. Since the full data set is approximately used by the model on the right plot, only few predictors got importance but the left plot used partial dataset to get importance over few more predictors.
2. Learning Rate : It means a higher number of predictions are added to the model output. Since the right hand plot has more predictions the correlation is more and hence only the first few predictors were considered significant.

Question

(b) Which model do you think would be more predictive of other samples?

Response

Because of the above explanations the two parameters are crucial in selecting the more predictive model. In my opinion the model with lesser fraction and learning rate would be more predictive than the one with lower values. Hence the left hand plot is more predictive.

Question

(c) How would increasing interaction depth affect the slope of predictor importance for either model in Fig. 8.24?

Code

|  |
| --- |
| X=pd.read\_csv(‘https://raw.githubusercontent.com/jey1987/DATA-624/main/HW2/solTrainXtrans.csv')  y=pd.read\_csv('https://raw.githubusercontent.com/jey1987/DATA-624/main/HW2/solTrainY.csv')  X\_matrix = np.array(X) y = np.array(y) model = GradientBoostingRegressor(max\_depth=1) model.fit(X\_matrix, y) # get importance importance = model.feature\_importances\_ # summarize feature importance df = pd.DataFrame(columns=['feature','importance']) cols=X.columns[1:229] cols rows\_list = [] for i,v in enumerate(importance):  dic1={}  feature\_res=cols[i-1]  importance\_res=v  dic1.update({'feature':feature\_res,'importance':v})  rows\_list.append(dic1) df=df.append(rows\_list) df\_result\_1=df.sort\_values(by='importance', ascending=False).head(10) model = GradientBoostingRegressor(max\_depth=10) model.fit(X\_matrix, y) # get importance importance = model.feature\_importances\_ # summarize feature importance  df = pd.DataFrame(columns=['feature','importance']) cols=X.columns[1:229] cols rows\_list = [] for i,v in enumerate(importance):  dic1={}  feature\_res=cols[i-1]  importance\_res=v  dic1.update({'feature':feature\_res,'importance':v})  rows\_list.append(dic1) df=df.append(rows\_list) df\_result\_10=df.sort\_values(by='importance', ascending=False).head(10) |

Importance for Interaction Depth = 1

feature importance

219 NumCarbon 0.298746

0 SurfaceArea2 0.200081

209 MolWeight 0.140153

228 SurfaceArea2 0.111085

223 NumChlorine 0.078732

217 NumAromaticBonds 0.069416

227 SurfaceArea1 0.022801

226 HydrophilicFactor 0.021351

172 FP172 0.015762

44 FP044 0.007473

Importance for Interaction Depth = 10

feature importance

219 NumCarbon 0.373211

0 SurfaceArea2 0.269949

228 SurfaceArea2 0.149564

209 MolWeight 0.072544

226 HydrophilicFactor 0.041144

211 NumNonHAtoms 0.033300

214 NumMultBonds 0.012246

212 NumBonds 0.009590

227 SurfaceArea1 0.007554

221 NumOxygen 0.005318

Response

Looking at two sets of Importance values It is evident that with the interaction depth 10 the importance is spread across several predictors. This makes for a good model if the predictions are accurate.

Market Basket Analysis

I am assigning one simple problem on market basket analysis / recommender systems.

Imagine 10000 receipts sitting on your table. Each receipt represents a transaction with items that were purchased. The receipt is a representation of stuff that went into a customer’s basket – and therefore ‘Market Basket Analysis’. That is exactly what the Groceries Data Set contains: a collection of receipts with each line representing 1 receipt and the items purchased. Each line is called a transaction and each column in a row represents an item. Here is the dataset = GroceryDataSet.csv (comma separated file) You assignment is to use R to mine the data for association rules. You should report support, confidence and lift and your top 10 rules by lift.

Association Rules mining in R is achieved by usage of apriori function available under arules package. Association rules mining is a two step process.

1. Frequent Itemset Generation
2. Rules Generation

Once the rules are generated the top 10 rules can be classified based on the output. The below steps illustrate the step-by-step approach for mining association rules.

Question

Load and Summarize the dataset

Code

|  |
| --- |
| df = pd.read\_csv("https://raw.githubusercontent.com/jey1987/DATA-624/main/HW2/GroceryDataSet.csv")  transactions = [] for i in range(0, 9834):  transactions.append([str(df.values[i,j]) for j in range(0, 23)]) |

Response

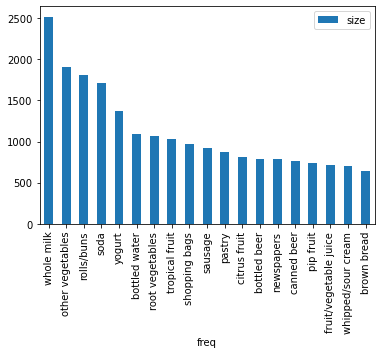
We load the data in with pandas and explore the raltio

Question

Find Frequently purchased products

Code

|  |
| --- |
| df = pd.DataFrame({'freq': sum(transactions,[])}) df\_freq=df.groupby('freq', as\_index=False).size().sort\_values(by="size",ascending=False).head(20).dropna() df\_freq=df\_freq[df\_freq['freq']!='nan'] df\_freq.plot.bar(x='freq',y='size') |



Reponse

Using a frequency plot it’s easy to see whole milk, other vegetables, and rolls appear the most in the data. We order the results by magnitude to easily visualize. Pandas has a plot method that allows for quick visualizations without the need for additional imports.

Question

**Association Rules generation:**

The association rules are generated using apriori function by passing the sparse matrix generated in the previous step.

**Choice of Minimum Support :**

Supp = (Products purchased at least 5 times a day)/total number of transactions

Supp = (6\*7)/nrow(df) = 0.004

**Choice of Confidence :**

After running trial and error on the different combinations the confidence of 0.3 was generating a decent number of rules as well as the combination.

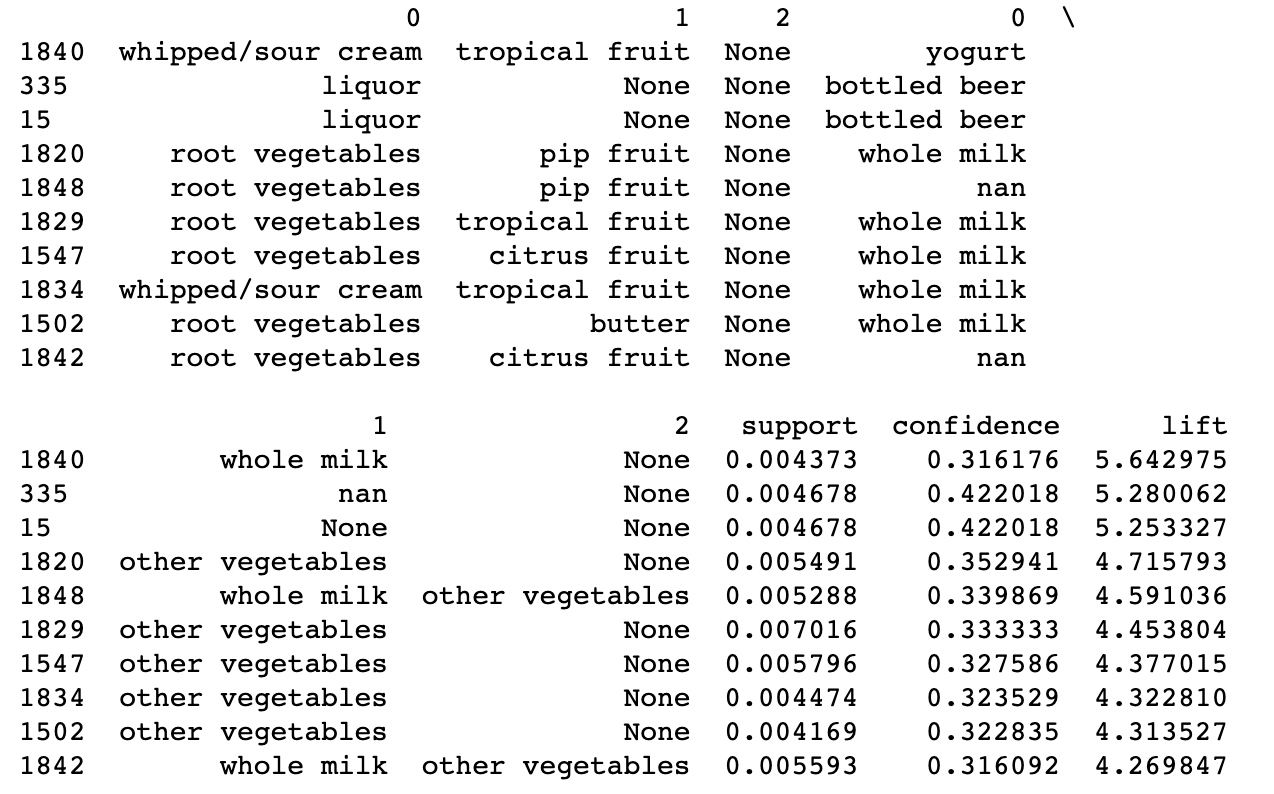
rule\_list = apriori(transactions, min\_support = 0.004, min\_confidence = 0.3)

**Top 10 Rules:**

Code

|  |
| --- |
| results = list(rule\_list) df\_results = pd.DataFrame(results) df\_results.head() support = df\_results.support print(len(results)) first\_values = [] second\_values = [] third\_values = [] fourth\_value = [] for i in range(df\_results.shape[0]):  single\_list = df\_results['ordered\_statistics'][i][0]  first\_values.append(list(single\_list[0]))  second\_values.append(list(single\_list[1]))  third\_values.append(single\_list[2])  fourth\_value.append(single\_list[3]) lhs = pd.DataFrame(first\_values) rhs= pd.DataFrame(second\_values) confidence=pd.DataFrame(third\_values,columns=['confidence']) lift=pd.DataFrame(fourth\_value,columns=['lift']) df\_final = pd.concat([lhs,rhs,support,confidence,lift], axis=1) df\_final.sort\_values(by='lift', ascending=False).head(10) |

[1] 1865



Response

From the results it looks like whipped cream and tropical fruit are often purchased together. We use pandas to reorder the data to where we can see the relationships.

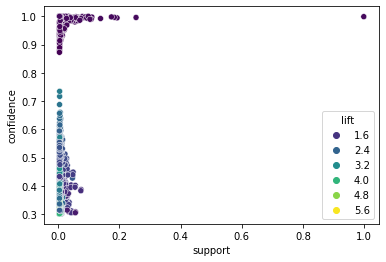
Question

Visualization of Association Rules

Scatter Plot

Code

|  |
| --- |
| scatter = sns.scatterplot(x="support", y="confidence",  hue="lift", # color dots by lift value  palette="viridis", # set colors  data=df\_final) |



Response

Here we see the scatterplot and the lift as the hue option to see how it is spread across the support and confidence. The plot here is particularly interesting because how low the support stays until confidence grows to 1.