CS 634 – Data Mining

Final Term Project

Name: Pooja Suthar

NJIT UCID: ps245

Email Address: ps245@njit.edu

Table of Contents

|  |  |
| --- | --- |
| 1. Overview | 3 |
| 1. System Requirements | 3 |
| 1. Dataset | 3 |
| 1. Usage 2. Code Information    1. Code for DecisionTree and NavieBayes 3. Output    * 1. Output 1(DecisionTree)      2. Output 2(Predicted Result)      3. Output 3(Classification Report)      4. Output 4(Heatmap)      5. Output 5(ROC curve)      6. Output 6(Comparison) | 4  4  5  9  9  10  15  17  18  20 |
| 1. Observations 2. References | 20  21 |
| 1. Appendix | 22 |

Overview:

This project is about implementing two Classification Algorithm:

* Decision Tree Algorithm - J48(C4.5)
* Navie Bayes Algorithm

Programming language used is Python.

System Requirements:

* Python 3.9.7
* pip

Dataset:

Data used for this project: [Bank Marketing Dataset](http://archive.ics.uci.edu/ml/machine-learning-databases/00222/)

* The above link will show two zip files. Download **bank.zip**.
* From bank.zip folder, I have used **bank.csv** file as my dataset.
* There is total 4522 rows in bank.csv file.

Bank-Marketing Dataset has 16 attributes and one class attribute.

Input variables:

1. age (numeric)
2. job : type of job (categorical: "admin." , "unknown", "unemployed", "management", "housemaid", "entrepreneur", "student", "blue-collar", "self-employed", "retired", "technician", "services")
3. marital : marital status (categorical: "married", "divorced", "single"; note: "divorced" means divorced or widowed)
4. education (categorical: "unknown", "secondary", "primary", "tertiary")
5. default: has credit in default? (binary : "yes", "no")
6. balance: average yearly balance, in euros (numeric)
7. housing: has housing loan? (binary: "yes", "no")
8. loan: has personal loan? (binary: "yes", "no")

# Related with the last contact of the current campaign:

1. contact: contact communication type (categorical: "unknown", "telephone", "cellular")
2. day: last contact day of the month (numeric)
3. month: last contact month of year (categorical: "jan", "feb", "mar", ..., "nov", "dec")
4. duration: last contact duration, in seconds (numeric)

# Other attributes:

1. campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)
2. pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric, -1 means client was not previously contacted)
3. previous: number of contacts performed before this campaign and for this client (numeric)
4. poutcome: outcome of the previous marketing campaign (categorical: "unknown", "other", "failure", "success")

Output variable (desired target):

1. class - has the client subscribed a term deposit? (binary: "yes", "no")

Usage:

* Create a folder. Inside the folder add a Jypter Notebook file containing the code given below.
* Add the dataset file to the same folder.
* Inside appendix, we have ‘requirement.txt’ file. Run “ pip install -r requirements.txt “ to download all library requirements.
* Run `<File\_name>.ipynb` .
* The Predicted Output is stored as csv file.
* Trained Decision Tree , Heatmaps and ROC curve of both algorithms are stored are png image.
* Classification report and Comparison of performance of both algorithms will be print on console.

Code Information:

Both Decision Tree and Naïve Bayes algorithms evaluating measures are same. Therefore, have created one file to remove duplicate code.

In the code, created different methods for each algorithm ; DecisionTree() and NavieBayes().

Each of this method calls different evaluating measure methods.

Code for Decision Tree and Naïve Bayes:

|  |
| --- |
| #Importing Libraries  **from** IPython**.**display **import** display  **import** seaborn **as** sns  **import** pandas **as** pd  **import** numpy **as** np  **from** sklearn **import** tree  **import** pydotplus  **import** matplotlib**.**pyplot **as** plt  **from** sklearn**.**tree **import** DecisionTreeClassifier  **from** sklearn**.**tree **import** plot\_tree  **from** sklearn**.**naive\_bayes **import** BernoulliNB  **from** sklearn**.**model\_selection **import** train\_test\_split**,**cross\_val\_score  **from** sklearn**.**metrics **import** confusion\_matrix**,** classification\_report**,** roc\_curve**,** roc\_auc\_score  **import** category\_encoders **as** ce  #Loading Dataset and declare feature vector and target variable  traindata**=**pd**.**read\_csv**(**"Traindata.csv"**)**  X **=** traindata**.**drop**([**'class'**],** axis**=**1**)** #feature vector  y **=** traindata**[**'class'**]** #target variable  #Encode categorical variables  **def** encoding**(**X\_train**,**X\_test**):**  encoder **=** ce**.**OneHotEncoder**(**cols**=[**'age'**,** 'job'**,** 'marital'**,** 'education'**,**'default'**,**'balance'**,**'housing'**,**'loan'**,**'contact'**,**'day'**,**'month'**,**'duration'**,**'campaign'**,**'pdays'**,**'previous'**,**'poutcome'**])**  X\_train **=** encoder**.**fit\_transform**(**X\_train**)**  X\_test **=** encoder**.**transform**(**X\_test**)**  **return** X\_train**,**X\_test    #Print the decision tree and store it as png image  **def** printTree**(**clf**,**X\_train**):**  cls**=[**'yes'**,**'no'**]**  dot\_data **=** tree**.**export\_graphviz**(**clf**,** out\_file**=None,**  feature\_names**=**X\_train**.**columns**,**  class\_names**=**cls**,**  filled**=True,** rounded**=True,**  special\_characters**=True)**  graph **=** pydotplus**.**graph\_from\_dot\_data**(**dot\_data**)**  graph**.**write\_png**(**'Decisiontree.png'**)**  #Create output csv file containing the predicted results  **def** printOutput**(**X\_test\_original**,**prediction**,**algoNum**):**  X\_test\_original**[**'Will subscribe a term deposit(yes/no)'**]=**prediction  pd**.**set\_option**(**'display.max\_rows'**,** **None)**  pd**.**set\_option**(**'display.max\_columns'**,** **None)**  pd**.**set\_option**(**'display.width'**,** 1000**)**  pd**.**set\_option**(**'display.colheader\_justify'**,** 'center'**)**  pd**.**set\_option**(**'display.precision'**,** 3**)**  **if(**algoNum **==** 1**):**  X\_test\_original**.**to\_csv**(**"DecisionTreeOutput.csv"**,**index**=False)**  **else:**  X\_test\_original**.**to\_csv**(**"NavieBayesOutput.csv"**,**index**=False)**    #Visualizing confusion matrix with seaborn heatmap and store output as png image  **def** printHeatMap**(**cm**,**algoNum**):**    cm\_matrix **=** pd**.**DataFrame**(**data**=**cm**,** columns**=[**'Actual Positive:1'**,**'Actual Negative:0'**],** index**=[**'Predict Positive:1'**,** 'Predict Negative:0'**])**  sns**.**heatmap**(**cm\_matrix**,** annot**=True,** fmt**=**'d'**,** cmap**=**'YlGnBu'**)**  **if(**algoNum **==** 1**):**  result\_path **=** 'DecisionTree\_HeatMap.png'  **else:**  result\_path**=**'NavieBayes\_HeatMap.png'  plt**.**savefig**(**result\_path**,** dpi**=**400**)**      #Created a function for calculating "Classification Accuracy Score","Classification Error", "Precision", "Recall", "F1 Score","True Positive Rate","False Positive Rate","True Negative Rate","False Negative Rate"  **def** ModelEvalution**(**y\_test**,**prediction**,**algoNum**):**  cm**=**confusion\_matrix**(**y\_test**,**prediction**)**  TP **=** cm**[**0**,**0**]**  TN **=** cm**[**1**,**1**]**  FP **=** cm**[**0**,**1**]**  FN **=** cm**[**1**,**0**]**  classification\_accuracy **=** **(**TP **+** TN**)** **/** **float(**TP **+** TN **+** FP **+** FN**)**  classification\_error **=** **(**FP **+** FN**)** **/** **float(**TP **+** TN **+** FP **+** FN**)**  precision **=** TP **/** **float(**TP **+** FP**)**  recall **=** TP **/** **float(**TP **+** FN**)**  f1\_score **=** 2 **\*** **(**precision **\*** recall**)** **/** **(**precision **+** recall**)**  true\_positive\_rate **=** TP **/** **float(**TP **+** FN**)**  false\_positive\_rate **=** FP **/** **float(**FP **+** TN**)**  true\_negative\_rate**=**TN **/** **float(**TN **+** FP**)**  false\_negative\_rate**=**FN **/** **float(**FN **+** TP**)**  specificity **=** TN **/** **(**TN **+** FP**)**  **if(**algoNum **==** 1**):**  printHeatMap**(**cm**,**1**)**  **else:**  printHeatMap**(**cm**,**2**)**  **return** **[round(**classification\_accuracy**,**3**),round(**classification\_error**,**3**),**precision**,**recall**,**f1\_score**,**true\_positive\_rate**,**false\_positive\_rate**,**true\_negative\_rate**,**false\_negative\_rate**]**  #Applying 10-Fold Cross Validation  **def** crossValidation\_Score**(**clf**,**X\_train**,**y\_train**):**  scores **=** cross\_val\_score**(**clf**,**X\_train**,**y\_train**,**cv **=** 10**,**scoring**=**'accuracy'**)**  cross\_validation\_score**=**scores**.**mean**()**  **return** **round(**scores**.**mean**(),**3**)**  #Plotting ROC Curve and computing Area Under Curve(AUC)  **def** ROC\_Curve**(**clf**,**X\_test**,**y\_test**,**algoNum**):**  y\_pred1 **=** clf**.**predict\_proba**(**X\_test**)[:,** 1**]**  fpr**,** tpr**,** thresholds **=** roc\_curve**(**y\_test**,** y\_pred1**,** pos\_label **=** 'yes'**)**  ROC\_AUC **=** roc\_auc\_score**(**y\_test**,** y\_pred1**)**  plt**.**figure**(**figsize**=(**6**,**4**))**  plt**.**plot**(**fpr**,** tpr**,** linewidth**=**2**)**  plt**.**xlabel**(**'False Positive Rate (Specificity)'**)**  plt**.**ylabel**(**'True Positive Rate (Sensitivity)'**)**  **if(**algoNum **==** 1**):**  plt**.**title**(**'ROC curve for Decision Classifier for Predicting Subscription'**)**  result\_path **=** 'DecisionTree\_ROCcurve.png'  **else:**  plt**.**title**(**'ROC curve for NavieBayes Classifier for Predicting Subscription')  result\_path**=**'NavieBayes\_ROCcurve.png'  plt**.**savefig**(**result\_path**,** dpi**=**400**)**  **return** ROC\_AUC  #Printing Classification Report of each algorithm  **def** print\_classification\_report(y\_test,prediction,algoNum):  **if**(algoNum == 1):  **print**("Classification Report of Decision Tree Algorithm:\n" , classification\_report(y\_test,prediction))  **else**:  **print**("Classification Report of Navie Bayes Algorithm:\n" , classification\_report(y\_test,prediction))  #Created a function for Decision Tree Classifier  **def** Decisiontree**():**  #Split data into separate training and test set  X\_train**,** X\_test**,** y\_train**,** y\_test**=**train\_test\_split**(**X**,**y**,**train\_size**=**0.8**,** test\_size**=**0.2**,** random\_state**=**10**)**  X\_test\_original**=**X\_test**.**copy**(**deep**=False)**  X\_train\_original**=**X\_train**.**copy**(**deep**=False)**  X\_train**,**X\_test**=**encoding**(**X\_train**,**X\_test**)**  clf **=** DecisionTreeClassifier **(**criterion**=**"entropy"**,** splitter**=**"best"**,** max\_depth **=** 4**,**random\_state**=**10**)**  #max\_depth is maximum number of levels in the tree  clf**.**fit**(**X\_train**,**y\_train**)**  printTree**(**clf**,**X\_train**)**  prediction**=**clf**.**predict**(**X\_test**)**  printOutput**(**X\_test\_original**,**prediction**,**1**)**  evalList**=**ModelEvalution**(**y\_test**,**prediction**,**1**)**  evalList**.**append**(**crossValidation\_Score**(**clf**,**X\_train**,**y\_train**))**  evalList**.**append**(**ROC\_Curve**(**clf**,**X\_test**,**y\_test**,**1**))**  evalList**.**insert**(**0**,round(**clf**.**score**(**X\_train**,**y\_train**),**3**))**  evalList**.**insert**(**1**,round(**clf**.**score**(**X\_test**,** y\_test**),**3**))**  print\_classification\_report(y\_test,prediction,1)  **return** evalList    #Created a function for Navie Bayes Classifier  **def** NaiveBayes**():**  X\_train**,** X\_test**,** y\_train**,** y\_test**=**train\_test\_split**(**X**,**y**,**train\_size**=**0.8**,** test\_size**=**0.2**,** random\_state**=**10**)**  X\_test\_original**=**X\_test**.**copy**(**deep**=False)**  X\_train\_original**=**X\_train**.**copy**(**deep**=False)**  X\_train**,**X\_test**=**encoding**(**X\_train**,**X\_test**)**  nv **=** BernoulliNB**()**  nv**.**fit**(**X\_train**,** y\_train**)**  y\_pred **=** nv**.**predict**(**X\_test**)**  printOutput**(**X\_test\_original**,**y\_pred**,**2**)**  evalList**=**ModelEvalution**(**y\_test**,**y\_pred**,**2**)**  evalList**.**append**(**crossValidation\_Score**(**nv**,**X\_train**,**y\_train**))**  evalList**.**append**(**ROC\_Curve**(**nv**,**X\_test**,**y\_test**,**2**))**  evalList**.**insert**(**0**,round(**nv**.**score**(**X\_train**,**y\_train**),**3**))**  evalList**.**insert**(**1**,round(**nv**.**score**(**X\_test**,** y\_test**),**3**))**  print\_classification\_report(y\_test, y\_pred ,2)  **return** evalList    #Calling Decisiontree function and NaiveBayes function  dtlist**=**Decisiontree**()**  nbList**=**NaiveBayes**()**  #Displaying the comparison between Decision tree and Naive Bayes  **print**("Comparing the performance of two Algorithms\n")  data **=** **{**'Decision Tree Algorithm'**:**dtlist**,**'Navie Bayes Algorithm'**:**nbList**}**  headers**=[**"Training set score"**,**"Test set score"**,**"Classification Accuracy Score"**,**"Classification Error"**,** "Precision"**,** "Recall"**,** "F1 Score"**,**"True Positive Rate"**,**"False Positive Rate"**,**"True Negative Rate"**,**"False Negative Rate"**,**"10 Fold Cross Validation Score"**,**"ROC\_AUC"**]**  **print(**pd**.**DataFrame**(**data**,** headers**))** |

Output:

1. Output of printTree() method:

Decision tree obtain after training phase stored as Decisiontree.png image.

Diagram

Description automatically generated

Root of the Tree : poutcome

Impurity Measure used: Entropy

Samples indicates number of rows of data left after this level.

Class=’yes’ means the customer will subscribe a term deposit

Class=’no’ means the customer will not subscribe a term deposit.

1. Output of printOutput() method:

Graphical user interface, table, Excel

Description automatically generatedThe predicted results of Decision Tree Classifier stored as DecisionTreeOutput.csv file :

The last column shows whether the person having these values of variables will subscribe a term deposit or not.

Graphical user interface, application, table, Excel

Description automatically generatedTable, Excel

Description automatically generated

Graphical user interface, application, table, Excel

Description automatically generated

The predicted Results of Navie Bayes Classifier stored as NavieBayesOutput.csv file:

Graphical user interface, table, Excel

Description automatically generatedGraphical user interface, application, table, Excel

Description automatically generated

Graphical user interface, application, table, Excel

Description automatically generatedGraphical user interface, application, table, Excel

Description automatically generated

1. Output of print\_classification\_report() method:

**Classification Report:**

It is used to measure the **quality of predictions from a classification algorithm** i.e. How many predictions are ‘yes’ and how many are ‘no’.

The report shows the main classification metrics precision, recall and f1-score on a per-class basis. The metrics are calculated by using true and false positives, true and false negatives.

**Precision** – What percent of the predictions are correct? i.e., the ability of a classifier not to label an instance positive that is actually negative. For each class it is defined as the ratio of true positives to the sum of true and false positives.

TP – True Positives

FP – False Positives

Precision – Accuracy of positive predictions.

Precision = TP/(TP + FP)

### ****Recall** – What percent of the positive cases did the algorithm catch? i.e.,** the ability of a classifier to find all positive instances. For each class it is defined as the ratio of true positives to the sum of true positives and false negatives.

**FN – False Negatives**

Recall: Fraction of positives that were correctly identified.  
Recall = TP/(TP+FN)

### ****F1** **score** – What percent of positive predictions were correct?**

The F1 score is a weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0.

F1 Score = 2\*(Recall \* Precision) / (Recall + Precision)

Classification Report of Decision Tree Algorithm:

A black screen with white text

Description automatically generated with low confidence

Classification Report of Navie Bayes Algorithm:

A picture containing text, black, screen, hand

Description automatically generated

From above classification report, Navie Bayes Algorithm has better report than Decision Tree Algorithm.

1. Output of printHeatMap() method:

**Heatmap:**

Heatmap is a visualization that displays data in a color encoded matrix.

It represents how many actual positive examples are predicted positive and how many actual negative examples are predicted negative.

Heatmap of Decision Tree stored as DecisionTree\_HeatMap.png:

Chart, treemap chart

Description automatically generated

From the image, it is concluded that there is total 880 negative examples out of which 789 examples are predicted negative (correctly) and 95 examples are predicted positive (incorrectly).

Similarly, there are total 25 positive examples out of which 11 examples are predicted negative (incorrect) and 14 examples are predicted positive (correctly).

Heatmap of Navie Bayes Algorithm stored as NavieBayes\_HeatMap.png image:

From the image, it is concluded that there is total 878 negative examples out of which 788 examples are predicted negative (correctly) and 90 examples are predicted positive (incorrectly).

Similarly, there are total 27 positive examples out of which 8 examples are predicted negative (incorrect) and 19 examples are predicted positive (correctly).

Chart, treemap chart

Description automatically generated

1. Output of ROC\_Curve() method:

**ROC Curve:**

**A Receiver Operator Characteristic (ROC) curve is a graphical plot used to show the diagnostic ability of binary classifiers.**

A ROC curve is constructed by plotting the true positive rate (TPR) against the false positive rate (FPR).

The true positive rate is the proportion of observations that were correctly predicted to be positive out of all positive observations (TP/(TP + FN)).

Similarly, the false positive rate is the proportion of observations that are incorrectly predicted to be positive out of all negative observations (FP/(TN + FP))

The ROC curve shows the trade-off between sensitivity (or TPR) and specificity (1 – FPR).

* Classifiers that give curves closer to the top-left corner indicate a better performance.
* As a baseline, a random classifier is expected to give points lying along the diagonal (FPR = TPR). The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test.

ROC Curve of Decision Tree Classifier stored as DecisionTree\_ROCcurve.png:

Chart, line chart

Description automatically generated

ROC Curve of Navie Bayes Classifier stored as NavieBayes\_ROCcurve.png image:

Chart, line chart

Description automatically generated

From both curves, it is concluded that ROC curve of Naïve Bayes is more toward top-left corner showing better performance while ROC curve of Decision Tree is more towards diagonal (black dotted line), less accurate.

1. Output of ModelEvalution() method:

**Comparison of Performance of both the Algorithms:**

Text

Description automatically generated

Observations:

* By comparing the accuracy of the above two algorithms it is observed that the Naïve Bayes algorithm is better than decision tree algorithm.
* As per the 10 fold cross validation, **Naïve bayes algorithm gives 89.3%** accuracy and **Decision tree algorithm gives 89.4**% accuracy which is almost same**.**
* As per ROC\_AUC(Area Under Curve), **Naïve bayes algorithm gives 0.743 area** and **Decision tree algorithm gives 0.663 area.**
* There is total 906 rows in test data. By comparing the output of the test data:

1. Naïve Bayes algorithm gets 807/905 correct prediction
2. Decision tree algorithm gets 799/905 correct prediction

Reference of Code:

[Decision-Tree Classifier Tutorial | Kaggle](https://www.kaggle.com/code/prashant111/decision-tree-classifier-tutorial)

[Naive Bayes Classifier in Python | Kaggle](https://www.kaggle.com/code/prashant111/naive-bayes-classifier-in-python)

Appendix:

Requirement.txt file:

asttokens @ file:///home/conda/feedstock\_root/build\_artifacts/asttokens\_1618968359944/work

backcall @ file:///home/conda/feedstock\_root/build\_artifacts/backcall\_1592338393461/work

backports.functools-lru-cache @ <file:///home> /conda/feedstock\_root /build\_artifacts /backports.functools\_lru\_cache\_1618230623929/work

Bottleneck @ file:///C:/ci/bottleneck\_1648010904582/work

category-encoders @ file:///tmp/build/80754af9/category\_encoders\_1633413589380/work/dist

certifi==2021.10.8

cloudpickle @ file:///tmp/build/80754af9/cloudpickle\_1632508026186/work

colorama @ file:///home/conda/feedstock\_root/build\_artifacts/colorama\_1602866480661/work

cycler @ file:///tmp/build/80754af9/cycler\_1637851556182/work

cytoolz==0.11.0

daal4py==2021.5.0

dask @ file:///opt/conda/conda-bld/dask-core\_1647268715755/work

debugpy @ file:///D:/bld/debugpy\_1636043436560/work

decorator @ file:///home/conda/feedstock\_root/build\_artifacts/decorator\_1641555617451/work

distro @ file:///tmp/build/80754af9/distro\_1610479853857/work

entrypoints @ file:///home/conda/feedstock\_root/build\_artifacts/entrypoints\_1643888246732/work

executing @ file:///home/conda/feedstock\_root/build\_artifacts/executing\_1646044401614/work

fonttools==4.25.0

fsspec @ file:///opt/conda/conda-bld/fsspec\_1647268051896/work

future @ file:///C:/ci/future\_1607568713721/work

graphviz @ file:///tmp/build/80754af9/python-graphviz\_1612303637553/work

imagecodecs @ file:///C:/ci/imagecodecs\_1635511087451/work

imageio @ file:///tmp/build/80754af9/imageio\_1617700267927/work

ipykernel @ file:///D:/bld/ipykernel\_1647271331646/work/dist/ipykernel-6.9.2-py3-none-any.whl

ipython @ file:///D:/bld/ipython\_1646325032429/work

jedi @ file:///D:/bld/jedi\_1637175252639/work

joblib @ file:///tmp/build/80754af9/joblib\_1635411271373/work

jupyter-client @ file:///home/conda/feedstock\_root/build\_artifacts/jupyter\_client\_1642858610849/work

jupyter-core @ file:///D:/bld/jupyter\_core\_1645024665167/work

kiwisolver @ file:///C:/ci/kiwisolver\_1644962577370/work

locket @ file:///C:/ci/locket\_1647006279389/work

matplotlib @ file:///C:/ci/matplotlib-suite\_1647423638658/work

matplotlib-inline @ file:///home/conda/feedstock\_root/build\_artifacts/matplotlib-inline\_1631080358261/work

mkl-fft==1.3.1

mkl-random @ file:///C:/ci/mkl\_random\_1626186184308/work

mkl-service==2.4.0

munkres==1.1.4

nest-asyncio @ file:///home/conda/feedstock\_root/build\_artifacts/nest-asyncio\_1638419302549/work

networkx @ file:///opt/conda/conda-bld/networkx\_1647437648384/work

numexpr @ file:///C:/ci/numexpr\_1640704337920/work

numpy @ file:///C:/ci\_310/numpy\_and\_numpy\_base\_1643798589088/work

packaging @ file:///tmp/build/80754af9/packaging\_1637314298585/work

pandas @ file:///C:/ci/pandas\_1647936849302/work

parso @ file:///home/conda/feedstock\_root/build\_artifacts/parso\_1638334955874/work

partd @ file:///opt/conda/conda-bld/partd\_1647245470509/work

patsy==0.5.2

pickleshare @ file:///home/conda/feedstock\_root/build\_artifacts/pickleshare\_1602536217715/work

Pillow==9.0.1

prompt-toolkit @ file:///home/conda/feedstock\_root/build\_artifacts/prompt-toolkit\_1644497866770/work

psutil @ file:///D:/bld/psutil\_1640887281543/work

pure-eval @ file:///home/conda/feedstock\_root/build\_artifacts/pure\_eval\_1642875951954/work

pydot @ file:///C:/ci/pydot\_1615395399420/work

pydotplus @ file:///tmp/build/80754af9/pydotplus\_1594390100614/work

Pygments @ file:///home/conda/feedstock\_root/build\_artifacts/pygments\_1641580240686/work

pyparsing @ file:///tmp/build/80754af9/pyparsing\_1635766073266/work

python-dateutil @ file:///tmp/build/80754af9/python-dateutil\_1626374649649/work

pytz==2021.3

PyWavelets @ file:///C:/ci/pywavelets\_1607645631519/work

pywin32==303

PyYAML==6.0

pyzmq @ file:///D:/bld/pyzmq\_1635877415276/work

scikit-build @ file:///C:/ci/scikit-build\_1610399389004/work

scikit-image==0.18.3

scikit-learn @ file:///C:/ci/scikit-learn\_1642617276183/work

scikit-learn-intelex==2021.20220215.102710

scikit-rf @ file:///tmp/build/80754af9/scikit-rf\_1624298598240/work

scipy @ file:///C:/ci/scipy\_1641555170412/work

seaborn @ file:///tmp/build/80754af9/seaborn\_1629307859561/work

sip==4.19.13

six @ file:///tmp/build/80754af9/six\_1644875935023/work

stack-data @ file:///home/conda/feedstock\_root/build\_artifacts/stack\_data\_1644872665635/work

statsmodels==0.13.2

threadpoolctl @ file:///Users/ktietz/demo/mc3/conda-bld/threadpoolctl\_1629802263681/work

tifffile @ file:///tmp/build/80754af9/tifffile\_1627275862826/work

toolz @ file:///tmp/build/80754af9/toolz\_1636545406491/work

tornado @ file:///C:/ci/tornado\_1606924294691/work

traitlets @ file:///home/conda/feedstock\_root/build\_artifacts/traitlets\_1635260543454/work

wcwidth @ file:///home/conda/feedstock\_root/build\_artifacts/wcwidth\_1600965781394/work

wincertstore==0.2