KAMRAN BALAYEV DATA MINING SEMESTER PROJECT

Climate Model Simulation Crashes

This dataset contains records of simulation crashes encountered during climate model uncertainty quantification ensembles. Ensemble members were constructed using a Latin hypercube method in LLNL's UQ Pipeline software system to sample the uncertainties of 18 model parameters within the Parallel Ocean Program (POP2) component of the Community Climate System Model (CCSM4).

Three separate Latin hypercube ensembles were conducted, each containing 180 ensemble members. 46 out of the 540 simulations failed for numerical reasons at combinations of parameter values.

The goal is to use classification to predict simulation outcomes (fail or succeed) from input parameter values, and to use sensitivity analysis and feature selection to determine the causes of simulation crashes.

Attribute Information:

The goal is to predict climate model simulation outcomes (column 21, fail or succeed) given scaled values of climate model input parameters (columns 3-20).

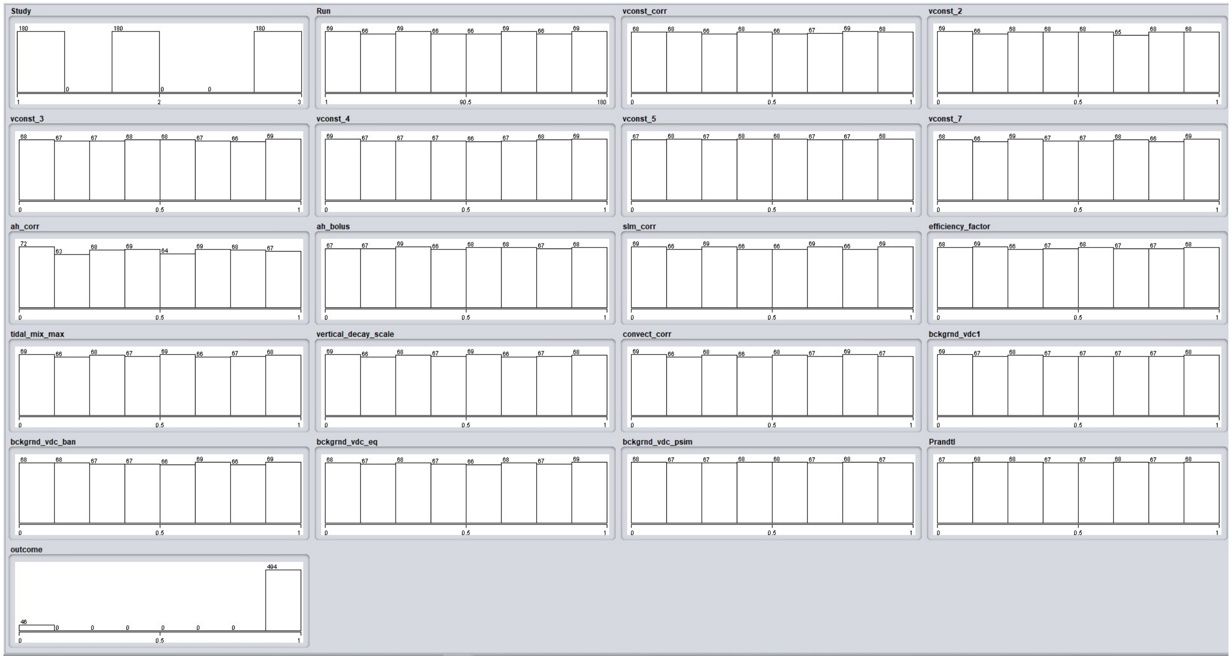
Column 1: Latin hypercube study ID (study 1 to study 3)

Column 2: simulation ID (run 1 to run 180)

Columns 3-20: values of 18 climate model parameters scaled in the interval [0, 1]

Column 21: simulation outcome (0 = failure, 1 = success)

Visualization of Classes via Usage of Weka



Missing Data Results

Weka tool represents that missing data rates are zero for all of the attributes

Name: Study Type: Numeric
Missing: 0 (0%) Distinct: 3 Unique: 0 (0%)

Name: Run Type: Numeric
Missing: 0 (0%) Distinct: 180 Unique: 0 (0%)

Name: vconst_corr Type: Numeric
Missing: 0 (0%) Distinct: 540 Unique: 540 (100%)

Name: vconst_2 Type: Numeric
Missing: 0 (0%) Distinct: 540 Unique: 540 (100%)

Name: vconst_3 Type: Numeric
Missing: 0 (0%) Distinct: 540 Unique: 540 (100%)

Name: vconst_4 Type: Numeric
Missing: 0 (0%) Distinct: 540 Unique: 540 (100%)

Name: vconst_5 Type: Numeric
Missing: 0 (0%) Distinct: 540 Unique: 540 (100%)

Name: vconst_7 Type: Numeric
Missing: 0 (0%) Distinct: 540 Unique: 540 (100%)

Name: ah_corr Type: Numeric
Missing: 0 (0%) Distinct: 540 Unique: 540 (100%)

Name	Missing	Distinct	Type	Unique
ah_bolus	0 (0%)	540	Numeric	540 (100%)
slm_corr	0 (0%)	540	Numeric	540 (100%)
efficiency_factor	0 (0%)	540	Numeric	540 (100%)
tidal_mix_max	0 (0%)	540	Numeric	540 (100%)
vertical_decay_scale	0 (0%)	540	String	540 (100%)
convect_corr	0 (0%)	540	String	540 (100%)
bckgrnd_vdc1	0 (0%)	540	Numeric	540 (100%)
bckgrnd_vdc_ban	0 (0%)	540	Numeric	540 (100%)
bckgrnd_vdc_eq	0 (0%)	540	Numeric	540 (100%)
bckgrnd_vdc_psim	0 (0%)	540	Numeric	540 (100%)
Prandtl	0 (0%)	540	Numeric	540 (100%)
outcome	0 (0%)	2	Numeric	0 (0%)

In [140]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn import preprocessing
from sklearn.metrics import accuracy_score
```

In [141]:
```
data = pd.read_csv('data.csv')
data.head()
```

Out[141]:

	Study	Run	vconst_corr	vconst_2	vconst_3	vconst_4	vconst_5	vconst_7	ah_corr	ah_bolu
0	1	1	0.859036	0.927825	0.252866	0.298838	0.170521	0.735936	0.428325	0.56794
1	1	2	0.606041	0.457728	0.359448	0.306957	0.843331	0.934851	0.444572	0.82801
2	1	3	0.997600	0.373238	0.517399	0.504993	0.618903	0.605571	0.746225	0.19592
3	1	4	0.783408	0.104055	0.197533	0.421837	0.742056	0.490828	0.005525	0.39212
4	1	5	0.406250	0.513199	0.061812	0.635837	0.844798	0.441502	0.191926	0.48754

5 rows × 21 columns

In [142]:
```
data.shape
```

Out[142]:
```
(540, 21)
```

In [143]:
```
data.columns
```

Out[143]:
```
Index(['Study', 'Run', 'vconst_corr', 'vconst_2', 'vconst_3', 'vconst_4',
       'vconst_5', 'vconst_7', 'ah_corr', 'ah_bolus', 'slm_corr',
       'efficiency_factor', 'tidal_mix_max',
       'vertical_decay_scale            ', 'convect_corr', 'bckgrnd_vdc1',
       'bckgrnd_vdc_ban', 'bckgrnd_vdc_eq', 'bckgrnd_vdc_psim', 'Prandtl',
       'outcome'],
      dtype='object')
```

In [144]:
```
data.info
```

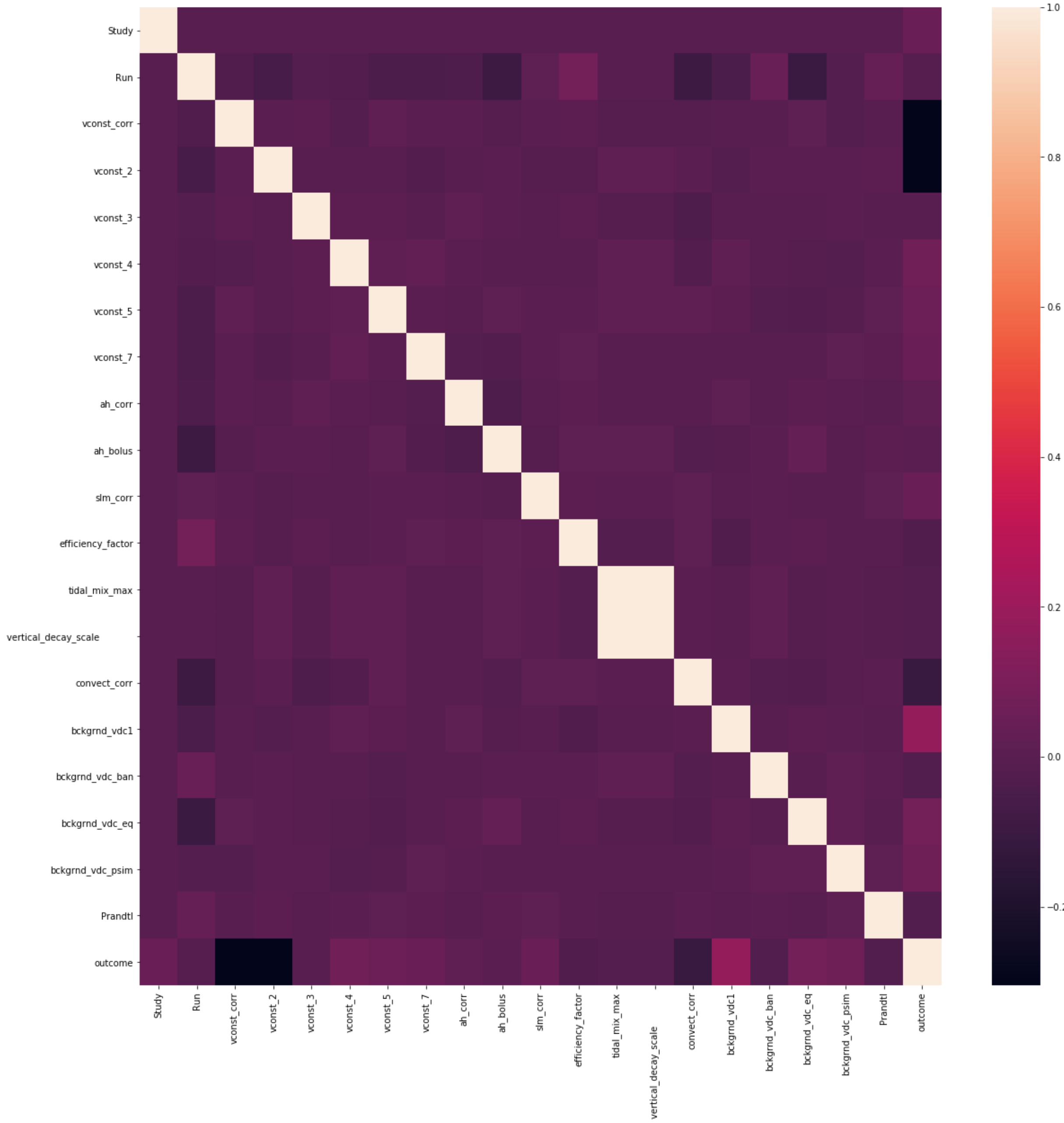
Out[144]:
```
<bound method DataFrame.info of      Study  Run  vconst_corr  vconst_2  vconst_
3  vconst_4  vconst_5  \
0        1    1     0.859036  0.927825  0.252866  0.298838  0.170521
1        1    2     0.606041  0.457728  0.359448  0.306957  0.843331
2        1    3     0.997600  0.373238  0.517399  0.504993  0.618903
3        1    4     0.783408  0.104055  0.197533  0.421837  0.742056
4        1    5     0.406250  0.513199  0.061812  0.635837  0.844798
..     ...  ...          ...       ...       ...       ...       ...
535      3  176     0.657136  0.489375  0.133713  0.411950  0.087780
536      3  177     0.915894  0.842720  0.518947  0.090622  0.336981
537      3  178     0.478600  0.941185  0.769245  0.950776  0.189406
538      3  179     0.007793  0.779287  0.867468  0.704820  0.983282
539      3  180     0.608075  0.031556  0.598264  0.794771  0.145680

     vconst_7   ah_corr  ah_bolus  ...  efficiency_factor  tidal_mix_max  \
0    0.735936  0.428325  0.567947  ...           0.245675       0.104226
1    0.934851  0.444572  0.828015  ...           0.616870       0.975786
2    0.605571  0.746225  0.195928  ...           0.679355       0.803413
3    0.490828  0.005525  0.392123  ...           0.471463       0.597879
4    0.441502  0.191926  0.487546  ...           0.551543       0.743877
..        ...       ...       ...  ...                ...            ...
535  0.356289  0.480204  0.029678  ...           0.280546       0.384117
536  0.893576  0.978703  0.674868  ...           0.798108       0.353546
537  0.112743  0.745645  0.527096  ...           0.193103       0.829563
538  0.420303  0.710612  0.174746  ...           0.761134       0.436714
539  0.378183  0.461948  0.425291  ...           0.480938       0.307816

     vertical_decay_scale           convect_corr  bckgrnd_vdc1  \
0                        0.104226      0.997518      0.448620
1                        0.975786      0.845247      0.864152
2                        0.803413      0.718441      0.924775
3                        0.597879      0.362751      0.912819
4                        0.743877      0.650223      0.522261
..                            ...           ...           ...
535                      0.384117      0.885948      0.459479
536                      0.353546      0.044796      0.347027
537                      0.829563      0.101506      0.381966
538                      0.436714      0.690132      0.981656
539                      0.307816      0.231638      0.583558

     bckgrnd_vdc_ban  bckgrnd_vdc_eq  bckgrnd_vdc_psim   Prandtl  outcome
0           0.307522        0.858310          0.796997  0.869893        0
1           0.346713        0.356573          0.438447  0.512256        1
2           0.315371        0.250642          0.285636  0.365858        1
3           0.977971        0.845921          0.699431  0.475987        1
4           0.043545        0.376660          0.280098  0.132283        1
..               ...             ...               ...       ...      ...
535         0.334482        0.573002          0.610183  0.737706        1
536         0.512499        0.810549          0.593332  0.142565        0
537         0.198811        0.867108          0.461632  0.652817        1
538         0.113193        0.364799          0.201469  0.536535        1
539         0.969365        0.464331          0.760344  0.762439        1

[540 rows x 21 columns]>
```

Visualization

In [145]:
```
data.hist(bins=60, figsize=(20,20))
plt.show()
```

Correlation Heatmap

In [146]:
```
plt.figure(figsize=(20,20))
sns.heatmap(data.corr())
plt.show()
```



In [147]:
```
data.describe()
```

Out[147]:

	Study	Run	vconst_corr	vconst_2	vconst_3	vconst_4	vconst_5	vco
count	540.000000	540.000000	540.000000	540.000000	540.000000	540.000000	540.000000	540.0(
mean	2.000000	90.500000	0.500026	0.500097	0.500027	0.500119	0.500001	0.4!
std	0.817254	52.008901	0.288939	0.288922	0.289067	0.288993	0.288827	0.2
min	1.000000	1.000000	0.000414	0.001922	0.001181	0.001972	0.000858	0.0(
25%	1.000000	45.750000	0.249650	0.251597	0.251540	0.250158	0.250630	0.2
50%	2.000000	90.500000	0.499998	0.499595	0.500104	0.500456	0.500903	0.4!
75%	3.000000	135.250000	0.750042	0.750011	0.749180	0.750348	0.748988	0.7
max	3.000000	180.000000	0.999194	0.998815	0.998263	0.997673	0.998944	0.9!

8 rows × 21 columns

In [148]:
```
#Control if there is empty spaces or not
data.isnull().sum()
```

Out[148]:
```
Study                   0
Run                     0
vconst_corr             0
vconst_2                0
vconst_3                0
vconst_4                0
vconst_5                0
vconst_7                0
ah_corr                 0
ah_bolus                0
slm_corr                0
efficiency_factor       0
tidal_mix_max           0
vertical_decay_scale    0
convect_corr            0
bckgrnd_vdc1            0
bckgrnd_vdc_ban         0
bckgrnd_vdc_eq          0
bckgrnd_vdc_psim        0
Prandtl                 0
outcome                 0
dtype: int64
```

Mix data set

In [149]:
```
data = data.sample(frac=1).reset_index(drop=True)
data.head()
```

Out[149]:

	Study	Run	vconst_corr	vconst_2	vconst_3	vconst_4	vconst_5	vconst_7	ah_corr	ah_bolu
0	2	71	0.390910	0.481451	0.729442	0.794123	0.097321	0.079587	0.064292	0.09317
1	1	163	0.062949	0.206038	0.114978	0.574029	0.482377	0.342907	0.553145	0.75796
2	3	42	0.103936	0.573524	0.095015	0.719870	0.419416	0.907753	0.916851	0.80122
3	3	89	0.162728	0.504645	0.656027	0.426580	0.939808	0.825156	0.349173	0.07902
4	2	153	0.857535	0.395180	0.776484	0.856685	0.224195	0.903020	0.669577	0.37370

5 rows × 21 columns

In [150]:
```
#get outcome column from dataframe
y = data['outcome']
X = data.drop('outcome',axis=1)
X.shape, y.shape
```

Out[150]: ((540, 20), (540,))

In [151]:
```
# Split data set 70 train 30 test
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size = 0.3, random_state
X_train.shape, X_test.shape
```

Out[151]: ((378, 20), (162, 20))

Classification Section

In [152]:
```
#theses lists will store the results of classification algorithms
model = []
trainAcc = []
testAcc = []

#function in order to store model and accuracy of it

def storeResults(MODEL, a,b):
    model.append(MODEL)
    trainAcc.append(round(a, 3))
    testAcc.append(round(b, 3))
```

Decision Tree Classifier

In [153]:
```
# Decision Tree model
from sklearn.tree import DecisionTreeClassifier

# instantiate the model with depth of 5
tree = DecisionTreeClassifier(max_depth = 5)
# fit the model
tree.fit(X_train, y_train)
```

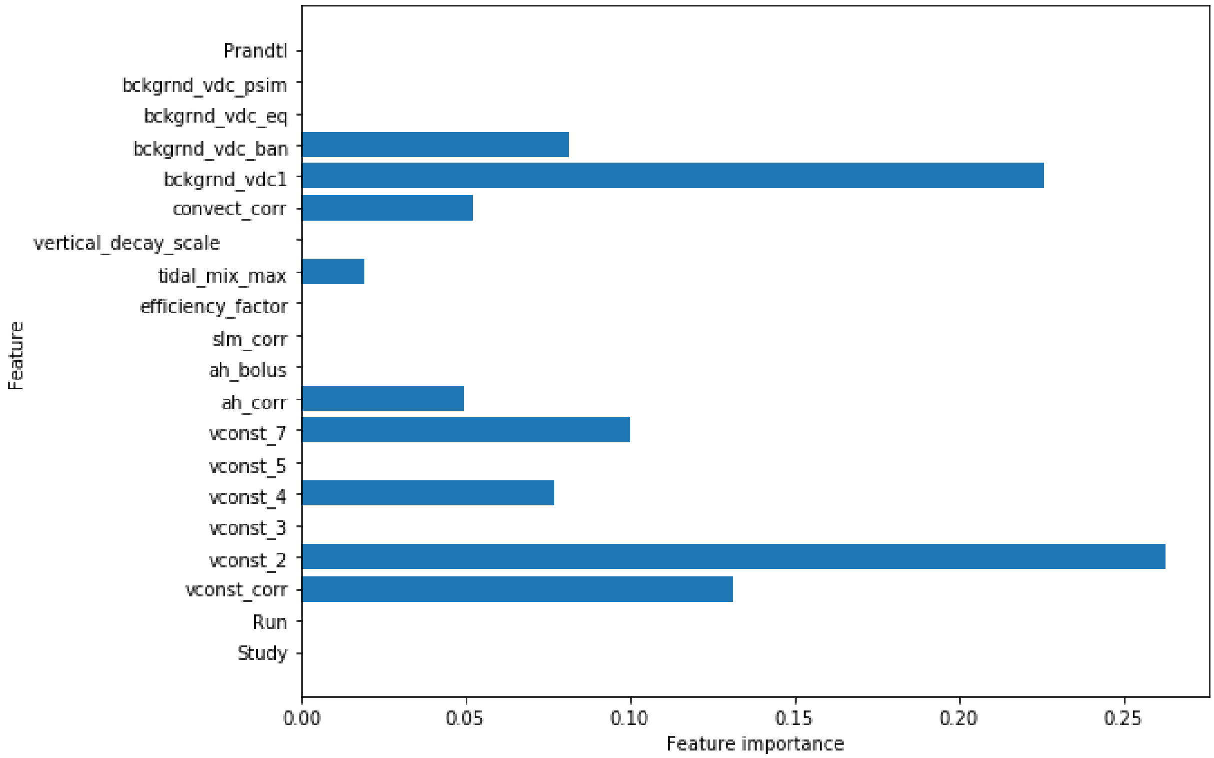
Out[153]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=5, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

In [154]:
```
##assign prediction results to the variables
yTestTree = tree.predict(X_test)
yTrainTree = tree.predict(X_train)
```

In [155]:
```
#Accuracy informations
acc_train_tree = accuracy_score(y_train,yTrainTree)
acc_test_tree = accuracy_score(y_test,yTestTree)

print("Decision Tree: Accuracy on training Data: {:.3f}".format(acc_train_tree))
print("Decision Tree: Accuracy on test Data: {:.3f}".format(acc_test_tree))
```

```
Decision Tree: Accuracy on training Data: 0.981
Decision Tree: Accuracy on test Data: 0.926
```

In [156]:
```
#Feature Importance
plt.figure(figsize=(9,7))
n_features = X_train.shape[1]
plt.barh(range(n_features), tree.feature_importances_, align='center')
plt.yticks(np.arange(n_features), X_train.columns)
plt.xlabel("Feature importance")
plt.ylabel("Feature")
plt.show()
```



In [157]:
```
#Store the accuracy result
storeResults('Decision Tree', acc_train_tree, acc_test_tree)
```

Random Forest Classifier

In [158]:
```
# Random Forest model
from sklearn.ensemble import RandomForestClassifier

# instantiate the model
forest = RandomForestClassifier(max_depth=5)

# fit the model
forest.fit(X_train, y_train)
```

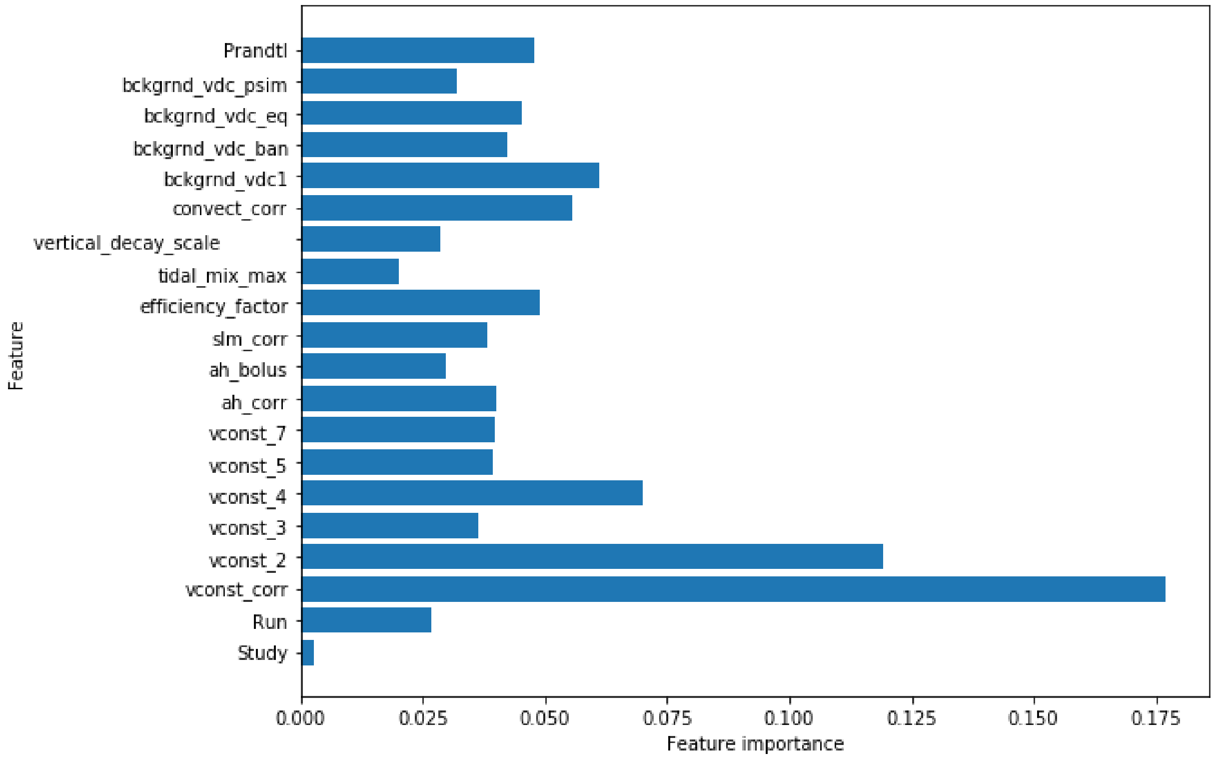
Out[158]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=5, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [159]:
```
##assign prediction results to the variables
y_test_forest = forest.predict(X_test)
y_train_forest = forest.predict(X_train)
```

In [160]:
```
#Accuracy informations
acc_train_forest = accuracy_score(y_train,y_train_forest)
acc_test_forest = accuracy_score(y_test,y_test_forest)

print("Random forest: Accuracy on training Data: {:.3f}".format(acc_train_forest)
print("Random forest: Accuracy on test Data: {:.3f}".format(acc_test_forest))
```

```
Random forest: Accuracy on training Data: 0.958
Random forest: Accuracy on test Data: 0.914
```

In [161]:
```
#Feature Importance
plt.figure(figsize=(9,7))
n_features = X_train.shape[1]
plt.barh(range(n_features), forest.feature_importances_, align='center')
plt.yticks(np.arange(n_features), X_train.columns)
plt.xlabel("Feature importance")
plt.ylabel("Feature")
plt.show()
```



In [162]:
```
#Sonucu tut
storeResults('Random Forest', acc_train_forest, acc_test_forest)
```

Support Vector Machines

In [163]:
```
#Support vector machine model
from sklearn.svm import SVC

# instantiate the model
svm = SVC(kernel='linear', C=1.0, random_state=12)
#fit the model
svm.fit(X_train, y_train)
```

Out[163]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=12, shrinking=True, tol=0.001,
    verbose=False)
```

In [164]:
```
####assign prediction results to the variables
y_test_svm = svm.predict(X_test)
y_train_svm = svm.predict(X_train)
```

In [165]:
```
#Accuracy informations
acc_train_svm = accuracy_score(y_train,y_train_svm)
acc_test_svm = accuracy_score(y_test,y_test_svm)

print("SVM: Accuracy on training Data: {:.3f}".format(acc_train_svm))
print("SVM : Accuracy on test Data: {:.3f}".format(acc_test_svm))
```

```
SVM: Accuracy on training Data: 0.950
SVM : Accuracy on test Data: 0.951
```

In [166]:
```
#Store results
storeResults('SVM', acc_train_svm, acc_test_svm)
```

KNN

In [167]:
```
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(X_train,y_train)
y_pred = neigh.predict(X_test)
accuracy_score(y_test, y_pred)
from sklearn import metrics
metrics.accuracy_score(y_test, y_pred)*100
```

Out[167]:
```
88.27160493827161
```

In [168]:
```
##assign prediction results to the variables
y_test_knn=neigh.predict(X_test)
y_train_knn = neigh.predict(X_train)
```

In [169]:
```
#Accuracy information
trainAccKnn = accuracy_score(y_train,y_train_knn)
testAccKnn = accuracy_score(y_test,y_test_knn)

print("Random forest: Accuracy on training Data: {:.3f}".format(trainAccKnn))
print("Random forest: Accuracy on test Data: {:.3f}".format(testAccKnn))
```

```
Random forest: Accuracy on training Data: 0.923
Random forest: Accuracy on test Data: 0.883
```

In [170]:
```
#Store result
storeResults('KNN', trainAccKnn, testAccKnn)
```

Clustering Algorithms

K means Clustering

In [171]:
```
from sklearn.cluster import KMeans
kMean = KMeans(n_clusters=5)
kMean.fit(X_train, y_train)
```

Out[171]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=5, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

In [172]:
```
##assign prediction results to the variables
yTestKmeans=kMean.predict(X_test)
yTrainKmeans = kMean.predict(X_train)
```

In [173]:
```
#Performance of model
trainAccKmeans = accuracy_score(y_train,yTrainKmeans)
testAccKmeans = accuracy_score(y_test,yTestKmeans)

print("Random forest: Accuracy on training Data: {:.3f}".format(trainAccKmeans))
print("Random forest: Accuracy on test Data: {:.3f}".format(testAccKmeans))
```

```
Random forest: Accuracy on training Data: 0.169
Random forest: Accuracy on test Data: 0.253
```

In [174]:
```
#Store results
storeResults('Kmeans', trainAccKmeans, testAccKmeans)
```

Agglomerative Clustering

In [175]:
```
from sklearn.cluster import AgglomerativeClustering
aggCluster = AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage=
```

In [176]:
```
##assign prediction results to the variables
yTestAggCluster=aggCluster.fit_predict(X_test)
yTrainAggCluster = aggCluster.fit_predict(X_train)
```

In [177]:
```
#MPerformance of model
trainAccAggCluster = accuracy_score(y_train,yTrainAggCluster)
testAccAggCluster = accuracy_score(y_test,yTestAggCluster)

print("Random forest: Accuracy on training Data: {:.3f}".format(trainAccAggCluste
print("Random forest: Accuracy on test Data: {:.3f}".format(testAccAggCluster))
```

```
Random forest: Accuracy on training Data: 0.169
Random forest: Accuracy on test Data: 0.253
```

In [178]:
```
#SStore result
storeResults('Agglomerative Clustering', trainAccAggCluster, testAccAggCluster)
```

Mean Shift Clustering

In [179]:
```
from sklearn.cluster import MeanShift, estimate_bandwidth
ms = MeanShift()
ms.fit(X)
```

Out[179]:
```
MeanShift(bandwidth=None, bin_seeding=False, cluster_all=True, max_iter=300,
          min_bin_freq=1, n_jobs=None, seeds=None)
```

In [180]:
```
##assign prediction results to the variables
yTestMs=ms.predict(X_test)
yTrainMs = ms.predict(X_train)
```

In [181]:
```
#Performance of model
trainAccMs = accuracy_score(y_train,yTrainMs)
testAccMs = accuracy_score(y_test,yTestMs)

print("Random forest: Accuracy on training Data: {:.3f}".format(trainAccMs))
print("Random forest: Accuracy on test Data: {:.3f}".format(testAccMs))
```

```
Random forest: Accuracy on training Data: 0.172
Random forest: Accuracy on test Data: 0.198
```

In [182]:
```
#Store result
storeResults('Mean Shift Clustering', trainAccMs, testAccMs)
```

In [183]:
```
#Create dataframe in order to store all algorithms and their performances.
results = pd.DataFrame({ 'model': model,
    'Train Accuracy': trainAcc,
    'Test Accuracy': testAcc})
results
```

Out[183]:

	model	Train Accuracy	Test Accuracy
0	Decision Tree	0.981	0.926
1	Random Forest	0.958	0.914
2	SVM	0.950	0.951
3	KNN	0.923	0.883
4	Kmeans	0.169	0.253
5	Agglomerative Clustering	0.169	0.253
6	Mean Shift Clustering	0.172	0.198

In [184]:
```
#Sort in  descending order
results.sort_values(by=['Test Accuracy', 'Train Accuracy'], ascending=False)
```

Out[184]:

	model	Train Accuracy	Test Accuracy
2	SVM	0.950	0.951
0	Decision Tree	0.981	0.926
1	Random Forest	0.958	0.914
3	KNN	0.923	0.883
4	Kmeans	0.169	0.253
5	Agglomerative Clustering	0.169	0.253
6	Mean Shift Clustering	0.172	0.198