```python
import numpy as np
import pandas as pd
import decimal
import seaborn as sns
import matplotlib.pyplot as plt
import copy
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import cohen_kappa_score
from sklearn.model_selection import learning_curve
from sklearn.model_selection import StratifiedKFold
from tqdm.notebook import tqdm
from yellowbrick.classifier import ROCAUC


def createModel(data, uniqueColumn, targetColumn, classifier):
    tDf = data.copy()

    # Get Y value from dataframe
    Y = np.array(tDf[targetColumn])

    # Drop unneeded columns for model training
    tDf.drop([uniqueColumn, targetColumn], axis=1, inplace=True)

    # Get X Value from rest of dataframe
    X = tDf.to_numpy()

    # fit model on training data
    model = copy.deepcopy(classifier)
    model.fit(X, Y)

    return model
```

```
37
38
39 def predictModel(model,
40                   data,
41                   uniqueColumn,
42                   targetColumn,
43                   colActual='y_test',
44                   colPredict='y_pred'):
45     tDf = data.copy()
46
47     Y = np.array(tDf[targetColumn])
48
49     # Drop unneeded columns for model testing
50     tDf.drop([uniqueColumn, targetColumn], axis=1,
   inplace=True)
51
52     # Get X Value from rest of dataframe
53     X = tDf.to_numpy()
54
55     y_pred = model.predict(X)
56
57     # make a dataframe for the results
58     tDf = pd.DataFrame(data=Y, columns=[colActual])
59     tDf[colPredict] = y_pred.tolist()
60
61     return tDf, colActual, colPredict
62
63
64 # Creates a dataframe with two columns:
65 # feature_idx: index of features
66 # importance: importance value of the relevent feature
67 def getModelFeatureImportance(model,
68                               featureLabel='feature_idx
   ',
69                               valueLabel='importance'):
70     # Create a dataframe with feature importances
71     impDf = pd.DataFrame(data=model.
   feature_importances_, columns=[valueLabel])
```

```
 72     impDf.reset_index(inplace=True)
 73     impDf.rename(columns={'index': featureLabel},
    inplace=True)
 74
 75     return impDf, featureLabel, valueLabel
 76
 77
 78 # plots importance of features in given model
 79 def analyzeModelFeatureImportance(data,
 80                                   valueLabel='
    importance',
 81                                   startValue=0.0001,
 82                                   increment=0.0001,
 83                                   upperValue=0.01,
 84                                   returnAbove=0.002,
 85                                   showSummary=True,
 86                                   showPlot=True):
 87     tqdm.pandas()
 88     xAxisLabel = 'Feature Importance'
 89     recCountLabel = 'Number of Documents'
 90     dx = startValue
 91
 92     # calc the number of decimals to round the value
 93     d = decimal.Decimal(str(increment))
 94     roundValue = d.as_tuple().exponent * -1
 95
 96     # Create the list with initial value of 0
 97     xAxisVal = [startValue]
 98
 99     while dx <= upperValue:
100         # add to the list of xAxisValues
101         xAxisVal.append(dx)
102
103         dx += increment
104         # round value included due to errors in FP
    addition
105         dx = round(dx, roundValue)
106
```

```
107     # turn list into dataframe
108     tDf = pd.DataFrame(xAxisVal, columns=[xAxisLabel])
109 
110     # Add in column for number of features >= that
    value
111     tDf[recCountLabel] = tDf.progress_apply(lambda x:
112                                             len(data.
    loc[data[valueLabel] >= x[xAxisLabel]]),
113                                             axis=1
114                                             )
115 
116     # return a list of features to be used in an
    optimized model
117     # it's by feature index
118     tDf2 = data.loc[data[valueLabel] >= returnAbove].
    copy()
119     tDf2.reset_index(drop=True, inplace=True)
120 
121     if showSummary:
122         # Give some sort of summary
123         indent = '---> '
124         print('Feature Importance Summary:')
125         print(f'{indent}Original feature count: {len(
    data)}')
126         print(f'{indent}Returned feature count: {len(
    tDf2)}')
127         print(f'{indent}Removed feature count: {len(
    data) - len(tDf2)}')
128         print(f'{indent}Return items above (including
    ): {returnAbove}')
129 
130     if showPlot:
131         # Plot it after the summary
132         tDf.plot(x=xAxisLabel,
133                  y=recCountLabel,
134                  ylabel='Cumulative count of documents
    ',
135                  title='Total Documents >= Importance
```

```
135 Level')
136 
137     return tDf2
138 
139 
140 def showAllModelFeatureImportance(data,
141                                   featureLabel,
142                                   valueLabel,
143                                   xlim=None):
144     #if len(data) > 100:
145     #    recLimit = 25
146     #else:
147     #    recLimit = max(len(data),25)
148 
149     recLimit = 25
150 
151     if xlim is None:
152         xlim = .03
153 
154     newFeatLabel = featureLabel + '_s'
155     tDf = data.sort_values(by=valueLabel, ascending=
    False).head(recLimit).copy()
156     tDf[newFeatLabel] = tDf.apply(lambda x:
157                                   'feature_' + str(x[
    featureLabel]),
158                                   axis=1
159                                   )
160 
161     sns.set_theme(style="whitegrid")
162 
163     # Initialize the matplotlib figure
164     f, ax = plt.subplots(figsize=(6, 10))
165 
166     # Plot the todtal crashes
167     # sns.set_color_codes("pastel")
168     sns.barplot(x=valueLabel,
169                 y=newFeatLabel,
170                 data=tDf,
```

```
171                 label="Total",
172                 palette="crest").set(title=f'Model
    Feature Importance (top {recLimit})')
173
174     plt.tick_params(axis='y', labelsize=10)
175
176     # ax.legend(ncol=1, loc="lower right", frameon=
    True)
177     ax.set(xlim=(0, xlim),
178            ylabel="",
179            xlabel="Feature Importance")
180     # sns.despine(left=True, bottom=True)
181     sns.despine()
182
183
184 def showConfusionMatrix(data,
185                         colNameActual,
186                         colNamePredict,
187                         axis_labels,
188                         titleSuffix,
189                         cmap='mako',
190                         plotsize=2):
191     plt.clf()
192     cm = confusion_matrix(np.array(pd.to_numeric(data[
    colNameActual])).reshape(-1, 1),
193                           np.array(pd.to_numeric(data[
    colNamePredict])).reshape(-1, 1)
194                           )
195
196     sns.heatmap(cm,
197                 annot=True,
198                 fmt='d',
199                 cmap=cmap,
200                 xticklabels=axis_labels,
201                 yticklabels=axis_labels
202                 )
203
204     if plotsize == 5:
```

```
205         sns.set(rc={'figure.figsize': (20, 8)})
206     elif plotsize == 4:
207         sns.set(rc={'figure.figsize': (15, 8)})
208     elif plotsize == 3:
209         sns.set(rc={'figure.figsize': (10, 8)})
210     elif plotsize == 2:
211         sns.set(rc={'figure.figsize': (8, 8)})
212     elif plotsize == 1:
213         sns.set(rc={'figure.figsize': (4, 8)})
214     else:  # Should be size 1
215         # should only be one but catch it and default
    to size 1
216         sns.set(rc={'figure.figsize': (4, 4)})
217 
218     # title with fontsize 20
219     plt.title(f'Confusion Matrix: {titleSuffix}',
    fontsize=20)
220     # x-axis label with fontsize 15
221     plt.xlabel('Predicted', fontsize=15)
222     # y-axis label with fontsize 15
223     plt.ylabel('Actual', fontsize=15)
224     plt.show()
225     plt.clf()
226 
227 
228 def showReport(data, colNameActual, colNamePredict,
    axisLabels, titleSuffix):
229     results = metrics.classification_report(pd.
    to_numeric(data[colNameActual]).to_list(),
230                                             data[
    colNamePredict].to_list(),
231 
    zero_division=0)
232     print(results)
233 
234     showConfusionMatrix(data=data,
235                         colNameActual=colNameActual,
236                         colNamePredict=colNamePredict,
```

```
237                          axis_labels=axisLabels,
238                          titleSuffix=titleSuffix
239                          )
240 
241 def showROCAUC(dataTrain,
242                dataTest,
243                classifier,
244                axisLabels,
245                colNameActual,
246                features):
247 
248     model = classifier
249     visualizer = ROCAUC(model, classes=axisLabels)
250 
251     # Fit the training data to the visualizer
252     visualizer.fit(dataTrain[features],
253                    dataTrain[colNameActual]
254                    )
255     # Evaluate model
256     visualizer.score(dataTest[features],
257                      dataTest[colNameActual]
258                      )
259 
260     visualizer.show()
261 
262     return visualizer
263 
264 def create_learning_curve(estimator,
265                           X,
266                           y,
267                           cv=None,
268                           n_jobs=None,
269                           train_sizes=None,
270                           verbose=4):
271     if train_sizes is None:
272         train_sizes = [0.1, 0.2, 0.5, 1.0]
273 
274     if cv is None:
```

```
275         cv = StratifiedKFold(n_splits=5, random_state=
    0, shuffle=True)
276 
277     train_sizes, train_scores, test_scores, fit_times
    , _ = learning_curve(
278         estimator,
279         X,
280         y,
281         cv=cv,
282         n_jobs=n_jobs,
283         train_sizes=train_sizes,
284         return_times=True,
285         verbose=verbose
286     )
287     return train_sizes, train_scores, test_scores,
    fit_times
288 
289 
290 def plot_learning_curve(train_sizes,
291                         train_scores,
292                         test_scores,
293                         fit_times,
294                         title,
295                         axes=None,
296                         ylim=None
297                         ):
298     """
299     Generate 3 plots: the test and training learning
    curve, the training
300     samples vs fit times curve, the fit times vs score
     curve.
301     """
302     plt.clf()
303 
304     if axes is None:
305         _, axes = plt.subplots(1, 3, figsize=(20, 5))
306 
307     axes[0].set_title(title)
```

```
308     if ylim is not None:
309         axes[0].set_ylim(*ylim)
310     axes[0].set_xlabel("Training examples")
311     axes[0].set_ylabel("Score")
312 
313     train_scores_mean = np.mean(train_scores, axis=1)
314     train_scores_std = np.std(train_scores, axis=1)
315     test_scores_mean = np.mean(test_scores, axis=1)
316     test_scores_std = np.std(test_scores, axis=1)
317     fit_times_mean = np.mean(fit_times, axis=1)
318     fit_times_std = np.std(fit_times, axis=1)
319 
320     # Plot learning curve
321     axes[0].grid()
322     axes[0].fill_between(
323         train_sizes,
324         train_scores_mean - train_scores_std,
325         train_scores_mean + train_scores_std,
326         alpha=0.1,
327         color="r",
328     )
329     axes[0].fill_between(
330         train_sizes,
331         test_scores_mean - test_scores_std,
332         test_scores_mean + test_scores_std,
333         alpha=0.1,
334         color="g",
335     )
336     axes[0].plot(
337         train_sizes, train_scores_mean, "o-", color="r
    ", label="Training score"
338     )
339     axes[0].plot(
340         train_sizes, test_scores_mean, "o-", color="g"
    , label="Cross-validation score"
341     )
342     axes[0].legend(loc="best")
343 
```

```
344     # Plot n_samples vs fit_times
345     axes[1].grid()
346     axes[1].plot(train_sizes, fit_times_mean, "o-")
347     axes[1].fill_between(
348         train_sizes,
349         fit_times_mean - fit_times_std,
350         fit_times_mean + fit_times_std,
351         alpha=0.1,
352     )
353     axes[1].set_xlabel("Training examples")
354     axes[1].set_ylabel("fit_times")
355     axes[1].set_title("Scalability of the model")
356 
357     # Plot fit_time vs score
358     fit_time_argsort = fit_times_mean.argsort()
359     fit_time_sorted = fit_times_mean[fit_time_argsort]
360     test_scores_mean_sorted = test_scores_mean[
    fit_time_argsort]
361     test_scores_std_sorted = test_scores_std[
    fit_time_argsort]
362     axes[2].grid()
363     axes[2].plot(fit_time_sorted,
    test_scores_mean_sorted, "o-")
364     axes[2].fill_between(
365         fit_time_sorted,
366         test_scores_mean_sorted -
    test_scores_std_sorted,
367         test_scores_mean_sorted +
    test_scores_std_sorted,
368         alpha=0.1,
369     )
370     axes[2].set_xlabel("fit_times")
371     axes[2].set_ylabel("Score")
372     axes[2].set_title("Performance of the model")
373 
374     plt.show()
375     plt.clf()
376 
```

```
377
378 def getModelAccuracy(data, colActual, colPredict):
379     accuracy = accuracy_score(data[colActual],
380                               data[colPredict])
381
382     return accuracy
383
384
385 def getModelPrecision(data, colActual, colPredict,
    average='weighted'):
386     precision = precision_score(data[colActual],
387                                 data[colPredict],
388                                 average=average)
389
390     return precision
391
392
393 def getModelRecall(data, colActual, colPredict,
    average):
394     recall = recall_score(data[colActual],
395                           data[colPredict],
396                           average=average)
397
398     return recall
399
400
401 def getModelF1(data, colActual, colPredict, average):
402     f1 = f1_score(data[colActual],
403                   data[colPredict],
404                   average=average)
405
406     return f1
407
408
409 def getModelCohenKappa(data, colActual, colPredict):
410     ck = cohen_kappa_score(data[colActual],
411                            data[colPredict])
412
```

```
413     return ck
414
```