Classification Assignment

February 22, 2019

1 Assignment 2 - Classification

2 Members - Amit Gandhi, Harshit Gupta, Karthik Venkata

In this assignment we will implement feature selection, cross validation and classification techniques. We will be using `sklearn` library to implement them. First we import the required libraries.

```
In [1]: import pandas as pd
        import numpy as np
        import seaborn as sns
        import time
        import itertools
        import matplotlib.pyplot as plt
        from sklearn.ensemble import ExtraTreesClassifier, RandomForestClassifier
        from mlxtend.feature_selection import SequentialFeatureSelector as sfs
        from sklearn.svm import LinearSVC, SVC
        from sklearn.model_selection import cross_validate
        from sklearn.model_selection import cross_val_score
        from sklearn import svm
        from sklearn.metrics import confusion_matrix,accuracy_score,precision_score,recall_sco
        from yellowbrick.classifier import ConfusionMatrix, ClassificationReport
        from yellowbrick.features import Rank2D
        import warnings
        warnings.filterwarnings('ignore')
```

We will be implementing the classification procedure in three phases: 1. Feature Selection 2. Model Selection 3. Model Evaluation

First, we get the training and test data.

```
In [2]: train = pd.read_csv('/Users/karthikjvn/Documents/Mac_Transfer_March2018/UIUC/Spring_20
```

```
In [3]: test = pd.read_csv('/Users/karthikjvn/Documents/Mac_Transfer_March2018/UIUC/Spring_201
```

We will now see the number of variables in our dataset.

```
In [4]: train.shape
```

Out[4]: (168, 148)

As we can see from the above ouput, our dataset has **148** variables.

In [5]: train.describe()

Out[5]:

	BrdIndx	Area	Round	Bright	Compact	\
count	168.000000	168.000000	168.000000	168.000000	168.000000	
mean	2.008512	565.869048	1.132976	165.569821	2.077679	
std	0.634807	679.852886	0.489150	61.883993	0.699600	
min	1.000000	10.000000	0.020000	37.670000	1.000000	
25%	1.537500	178.000000	0.787500	133.977500	1.547500	
50%	1.920000	315.000000	1.085000	164.485000	1.940000	
75%	2.375000	667.000000	1.410000	221.895000	2.460000	
max	4.190000	3659.000000	2.890000	244.740000	4.700000	

	ShpIndx	Mean_G	Mean_R	Mean_NIR	SD_G	...	\
count	168.000000	168.000000	168.000000	168.000000	168.000000	...	
mean	2.229881	161.577083	163.672440	171.459226	10.131369	...	
std	0.703572	63.407201	71.306748	67.973969	5.179409	...	
min	1.060000	30.680000	32.210000	40.120000	4.330000	...	
25%	1.700000	91.040000	101.187500	120.165000	6.770000	...	
50%	2.130000	187.560000	160.615000	178.345000	8.010000	...	
75%	2.680000	210.940000	234.815000	236.002500	11.500000	...	
max	4.300000	246.350000	253.080000	253.320000	36.400000	...	

	SD_NIR_140	LW_140	GLCM1_140	Rect_140	GLCM2_140	Dens_140	\
count	168.000000	168.000000	168.000000	168.000000	168.000000	168.000000	
mean	23.769881	3.098274	0.796488	0.665000	7.795536	1.594405	
std	12.836522	6.101883	0.103930	0.179086	0.670491	0.460627	
min	4.020000	1.000000	0.330000	0.240000	6.290000	0.230000	
25%	13.965000	1.395000	0.757500	0.560000	7.357500	1.325000	
50%	21.135000	1.740000	0.810000	0.690000	7.790000	1.660000	
75%	29.957500	2.285000	0.870000	0.810000	8.260000	1.945000	
max	60.020000	51.540000	0.950000	0.980000	9.340000	2.340000	

	Assym_140	NDVI_140	BordLngth_140	GLCM3_140
count	168.000000	168.000000	168.000000	168.000000
mean	0.615357	0.014583	983.309524	1275.292917
std	0.239900	0.153677	880.013745	603.658611
min	0.070000	-0.360000	56.000000	336.730000
25%	0.460000	-0.080000	320.000000	817.405000
50%	0.620000	-0.040000	776.000000	1187.025000
75%	0.810000	0.120000	1412.500000	1588.427500
max	1.000000	0.350000	6232.000000	3806.360000

[8 rows x 147 columns]

In [6]: first = train.head(5)

```
In [7]: first
```

Out[7]:

	class	BrdIndx	Area	Round	Bright	Compact	ShpIndx	Mean_G	Mean_R	\
0	car	1.27	91	0.97	231.38	1.39	1.47	207.92	241.74	
1	concrete	2.36	241	1.56	216.15	2.46	2.51	187.85	229.39	
2	concrete	2.12	266	1.47	232.18	2.07	2.21	206.54	244.22	
3	concrete	2.42	399	1.28	230.40	2.49	2.73	204.60	243.27	
4	concrete	2.15	944	1.73	193.18	2.28	4.10	165.98	205.55	

	Mean_NIR	...	SD_NIR_140	LW_140	GLCM1_140	Rect_140	GLCM2_140	\
0	244.48	...	26.18	2.00	0.50	0.85	6.29	
1	231.20	...	22.29	2.25	0.79	0.55	8.42	
2	245.79	...	15.59	2.19	0.76	0.74	7.24	
3	243.32	...	13.51	3.34	0.82	0.74	7.44	
4	208.00	...	15.65	50.08	0.85	0.49	8.15	

	Dens_140	Assym_140	NDVI_140	BordLngth_140	GLCM3_140
0	1.67	0.70	-0.08	56	3806.36
1	1.38	0.81	-0.09	1746	1450.14
2	1.68	0.81	-0.07	566	1094.04
3	1.36	0.92	-0.09	1178	1125.38
4	0.23	1.00	-0.08	6232	1146.38

[5 rows x 148 columns]

```
In [8]: last = train.tail(5)

In [9]: last
```

Out[9]:

	class	BrdIndx	Area	Round	Bright	Compact	ShpIndx	Mean_G	\
163	car	1.43	39	1.41	234.03	1.54	1.60	206.36	
164	soil	1.92	141	1.24	215.19	2.02	2.02	212.28	
165	grass	2.97	252	1.73	164.13	3.20	3.09	184.15	
166	grass	1.57	216	1.27	164.84	1.71	1.97	192.55	
167	concrete	2.12	836	0.88	232.84	1.78	2.52	202.39	

	Mean_R	Mean_NIR	...	SD_NIR_140	LW_140	GLCM1_140	Rect_140	\
163	246.05	249.69	...	55.92	1.73	0.65	0.81	
164	216.28	217.00	...	18.91	3.49	0.88	0.67	
165	152.03	156.22	...	33.52	2.02	0.86	0.71	
166	148.34	153.62	...	24.49	1.13	0.76	0.85	
167	247.24	248.89	...	7.84	1.52	0.76	0.24	

	GLCM2_140	Dens_140	Assym_140	NDVI_140	BordLngth_140	GLCM3_140
163	7.05	1.89	0.42	-0.10	66	2469.69
164	7.88	1.44	0.82	0.06	990	824.01
165	8.50	1.82	0.54	0.06	948	821.84
166	7.75	2.11	0.30	0.09	254	1580.72
167	7.16	0.74	0.49	-0.09	3020	1611.55

```
[5 rows x 148 columns]
```

We now look at the co-relation coefficients for all the variables in the dataset. We will be analyzing these co-efficients in batches.

```
In [10]: cm = sns.light_palette("green", as_cmap=True)
         df = train.corr().round(3).loc[:"GLCM3",:"GLCM3"]
         df = df.style.background_gradient(cmap=cm)
         print("Batch 1")
         df
```

```
Batch 1
```

```
Out[10]: <pandas.io.formats.style.Styler at 0x1098fb470>
```

```
In [11]: cm = sns.light_palette("blue", as_cmap=True)
         df = train.corr().round(3).loc["BrdIndx_40":"GLCM3_40",:"GLCM3"]
         df = df.style.background_gradient(cmap=cm)
         print("Batch 2")
         df
```

```
Batch 2
```

```
Out[11]: <pandas.io.formats.style.Styler at 0x1098f1ef0>
```

```
In [12]: cm = sns.light_palette("pink", as_cmap=True)
         df = train.corr().round(3).loc["BrdIndx_60":"GLCM3_60",:"GLCM3"]
         df = df.style.background_gradient(cmap=cm)
         print("Batch 3")
         df
```

```
Batch 3
```

```
Out[12]: <pandas.io.formats.style.Styler at 0x1a1dc81e10>
```

```
In [13]: cm = sns.light_palette("violet", as_cmap=True)
         df = train.corr().round(3).loc["BrdIndx_80":"GLCM3_80",:"GLCM3"]
         df = df.style.background_gradient(cmap=cm)
         print("Batch 4")
         df
```

```
Batch 4
```

```
Out[13]: <pandas.io.formats.style.Styler at 0x1a1dc876d8>
```

```
In [14]: cm = sns.light_palette("orange", as_cmap=True)
         df = train.corr().round(3).loc["BrdIndx_100":"GLCM3_100",:"GLCM3"]
         df = df.style.background_gradient(cmap=cm)
         print("Batch 5")
         df

Batch 5


Out[14]: <pandas.io.formats.style.Styler at 0x1a1e058a20>

In [15]: cm = sns.light_palette("red", as_cmap=True)
         df = train.corr().round(3).loc["BrdIndx_120":"GLCM3_120",:"GLCM3"]
         df = df.style.background_gradient(cmap=cm)
         print("Batch 6")
         df

Batch 6


Out[15]: <pandas.io.formats.style.Styler at 0x1a1dc925c0>

In [16]: cm = sns.light_palette("pink", as_cmap=True)
         df = train.corr().round(3).loc["BrdIndx_140":"GLCM3_140",:"GLCM3"]
         df = df.style.background_gradient(cmap=cm)
         print("Batch 7")
         df

Batch 7


Out[16]: <pandas.io.formats.style.Styler at 0x1a1dc6fd68>
```

2.1 Splitting the Dataset into Target and Predictor

```
In [17]: X_train = train.iloc[:,1:]
         Y_train = train.iloc[:,0]

In [18]: X_test = test.iloc[:,1:]
         Y_test = test.iloc[:,0]
```

2.2 Intuition behind Feature Selection method chosen and Procedure

By feature selection we delete redundant or meaningless features so that we can realize the higher generalization performance and faster classification than by the initial set of features.

In backward selection, we start from all the features and delete one feature at a time, which deteriorates the selection criterion the least. In forward selection, we start from an empty set of features and add one feature at a time, which improves the selection criterion the most.

We will be using two techniques to select the features in our dataset. They are :

- Step Forward Feature Selection using Random Forest Classifier.
- Using PCA for generating feature components.

sfs(Sequential Forward Selection) - We will define a classifier, as well as a step forward feature selector, and then perform our feature selection. The feature feature selector in mlxtend has some parameters we can define. The procedure involved in using sfs is -

1.First, we pass our classifier, the Random Forest classifier defined above the feature selector 2.Next, we define the subset of features we are looking to select (k_features=15) 3.We set the desired level of verbosity for mlxtend to report 4.Importantly, we set our scoring to accuracy; this is but one metric which could be used to score our resulting models built on the selected features 5.mlxtend feature selector uses cross validation internally, and we set our desired folds to 15 for our demonstration

We will use step forward feature selection. Here we will be using *accuracy* as our scoring metric.

Now we select the most important features.

2.3 Q.1 How did you choose which classification algorithm would be appropriate for this data?

Random Forest seemed to be appropriate for this particular classification task. Given the fact that we had data from images for which Support Vector Machine (SVM) is most effective due to which we assumed that we could use SVM. Kernels used in SVM include - linear and poly which are essentially different kinds of separating hyperplanes. Using SVM, we obtained an accuracy of about 60% for the linear case while Random forest gave 75%. In contrast, when 'poly' kernel was used, an accuracy of about 50% was obtained which seemed to be much lesser than what we got using Random Forest.

This is the intuition behind keeping both the models.

2.4 Bonus Task

As explained above, we used a SVM based classifier to compare our results with Random forest. The results were notably different and were much lower as compared with the Random Forest. Upon using the Linear kernel we got 60% whereas we obtained about 50% using Poly kernel. This is probably due to the fact that the number of instances available for classification were very low.In future, we can consider using oversampling to increase the size of the training dataset helping us to build a more accurate model.

3 Feature Selection

3.1 1. Random Forest based Feature Selection

Random Forest makes it very easy to measure the relative importance of each feature on the prediction.

3.2 CV = 5 (Number of folds for the cross-validation)

```
In [19]: clf1 = RandomForestClassifier()
```

```
        # Build step forward feature selection
        sfs1 = sfs(clf1,
                   k_features=15,
                   forward=True,
                   floating=False,
                   verbose=2,
                   scoring='accuracy',
                   cv=5)
```

```
In [20]: sfs1 = sfs1.fit(X_train, Y_train)
         feat_cols1 = list(sfs1.k_feature_idx_)
         print(feat_cols1)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 147 out of 147 | elapsed:   10.0s finished

[2019-02-22 19:57:15] Features: 1/15 -- score: 0.5341600882777353[Parallel(n_jobs=1)]: Using ba
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 146 out of 146 | elapsed:    8.4s finished

[2019-02-22 19:57:24] Features: 2/15 -- score: 0.7802096596214245[Parallel(n_jobs=1)]: Using ba
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 145 out of 145 | elapsed:    9.0s finished

[2019-02-22 19:57:33] Features: 3/15 -- score: 0.7842975978270095[Parallel(n_jobs=1)]: Using ba
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 144 out of 144 | elapsed:    8.8s finished

[2019-02-22 19:57:41] Features: 4/15 -- score: 0.8154469060351414[Parallel(n_jobs=1)]: Using ba
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 143 out of 143 | elapsed:    8.7s finished

[2019-02-22 19:57:50] Features: 5/15 -- score: 0.8334937611408201[Parallel(n_jobs=1)]: Using ba
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 142 out of 142 | elapsed:    7.8s finished

[2019-02-22 19:57:58] Features: 6/15 -- score: 0.8320125626007979[Parallel(n_jobs=1)]: Using ba
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 141 out of 141 | elapsed:    7.1s finished

[2019-02-22 19:58:05] Features: 7/15 -- score: 0.8524497071555895[Parallel(n_jobs=1)]: Using ba
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 140 out of 140 | elapsed:    6.7s finished

[2019-02-22 19:58:12] Features: 8/15 -- score: 0.845951956540192[Parallel(n_jobs=1)]: Using bac
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 139 out of 139 | elapsed:    6.8s finished
```

```
[2019-02-22 19:58:19] Features: 9/15 -- score: 0.8453458959341311[Parallel(n_jobs=1)]: Using ba
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 138 out of 138 | elapsed:    7.0s finished

[2019-02-22 19:58:26] Features: 10/15 -- score: 0.8513963161021983[Parallel(n_jobs=1)]: Using b
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 137 out of 137 | elapsed:    6.6s finished

[2019-02-22 19:58:32] Features: 11/15 -- score: 0.8385009761480349[Parallel(n_jobs=1)]: Using b
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 136 out of 136 | elapsed:    6.6s finished

[2019-02-22 19:58:39] Features: 12/15 -- score: 0.8609294626941686[Parallel(n_jobs=1)]: Using b
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 135 out of 135 | elapsed:    6.6s finished

[2019-02-22 19:58:45] Features: 13/15 -- score: 0.8499490705373057[Parallel(n_jobs=1)]: Using b
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 134 out of 134 | elapsed:    6.4s finished

[2019-02-22 19:58:52] Features: 14/15 -- score: 0.8560190136660724[Parallel(n_jobs=1)]: Using b
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s


[5, 18, 33, 38, 43, 48, 49, 60, 66, 80, 84, 92, 108, 128, 135]


[Parallel(n_jobs=1)]: Done 133 out of 133 | elapsed:    6.4s finished

[2019-02-22 19:58:58] Features: 15/15 -- score: 0.8697512944571768
```

3.3 CV = 10 (Number of folds for the cross-validation)

```
In [21]: clf2 = RandomForestClassifier() #Random Forest Implementation

         # Build step forward feature selection
         sfs2 = sfs(clf2,
                    k_features=15,
                    forward=True,
                    floating=False,
                    verbose=2,
                    scoring='accuracy',
                    cv=10)

In [22]: sfs2 = sfs2.fit(X_train, Y_train)

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
```

```
[Parallel(n_jobs=1)]: Done 147 out of 147 | elapsed:   14.7s finished

[2019-02-22 19:59:13] Features: 1/15 -- score: 0.5475518925518925[Parallel(n_jobs=1)]: Using ba
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 146 out of 146 | elapsed:   13.5s finished

[2019-02-22 19:59:26] Features: 2/15 -- score: 0.7647252747252746[Parallel(n_jobs=1)]: Using ba
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 145 out of 145 | elapsed:   14.2s finished

[2019-02-22 19:59:41] Features: 3/15 -- score: 0.8075763125763127[Parallel(n_jobs=1)]: Using ba
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 144 out of 144 | elapsed:   15.6s finished

[2019-02-22 19:59:56] Features: 4/15 -- score: 0.8455738705738707[Parallel(n_jobs=1)]: Using ba
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 143 out of 143 | elapsed:   14.4s finished

[2019-02-22 20:00:11] Features: 5/15 -- score: 0.8582600732600731[Parallel(n_jobs=1)]: Using ba
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 142 out of 142 | elapsed:   19.9s finished

[2019-02-22 20:00:31] Features: 6/15 -- score: 0.8585225885225884[Parallel(n_jobs=1)]: Using ba
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 141 out of 141 | elapsed:   14.6s finished

[2019-02-22 20:00:45] Features: 7/15 -- score: 0.8480586080586081[Parallel(n_jobs=1)]: Using ba
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 140 out of 140 | elapsed:   13.4s finished

[2019-02-22 20:00:59] Features: 8/15 -- score: 0.8565262515262516[Parallel(n_jobs=1)]: Using ba
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 139 out of 139 | elapsed:   13.2s finished

[2019-02-22 20:01:12] Features: 9/15 -- score: 0.8797863247863248[Parallel(n_jobs=1)]: Using ba
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 138 out of 138 | elapsed:   13.4s finished

[2019-02-22 20:01:25] Features: 10/15 -- score: 0.8894810744810744[Parallel(n_jobs=1)]: Using b
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 137 out of 137 | elapsed:   13.1s finished

[2019-02-22 20:01:38] Features: 11/15 -- score: 0.8902564102564103[Parallel(n_jobs=1)]: Using b
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 136 out of 136 | elapsed:   12.8s finished

[2019-02-22 20:01:51] Features: 12/15 -- score: 0.8846275946275945[Parallel(n_jobs=1)]: Using b
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
```

```
[Parallel(n_jobs=1)]: Done 135 out of 135 | elapsed:   13.4s finished

[2019-02-22 20:02:04] Features: 13/15 -- score: 0.880018315018315[Parallel(n_jobs=1)]: Using ba
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 134 out of 134 | elapsed:   12.7s finished

[2019-02-22 20:02:17] Features: 14/15 -- score: 0.8655860805860807[Parallel(n_jobs=1)]: Using b
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 133 out of 133 | elapsed:   13.1s finished

[2019-02-22 20:02:30] Features: 15/15 -- score: 0.8627167277167278
```

3.4 Subsetting the Training and Testing Dataset based on Feature Selection.

```
In [30]: feat_cols = list(sfs2.k_feature_idx_)
         cols = X_train.columns[feat_cols]

In [31]: X_train_final = pd.DataFrame(X_train.iloc[:,feat_cols])

In [32]: X_train_final.head()
```

Out[32]:

	Compact	SD_G	SD_NIR	SD_G_40	SD_R_40	GLCM3_40	SD_R_60	NDVI_60	\
0	1.39	21.41	18.69	27.63	28.36	3806.36	28.36	-0.08	
1	2.46	6.57	7.02	9.82	10.37	2581.63	19.62	-0.10	
2	2.07	6.16	5.53	6.55	7.01	3087.91	13.26	-0.07	
3	2.49	5.76	5.46	9.55	9.35	2194.40	10.36	-0.09	
4	2.28	11.46	9.77	11.46	8.90	2300.41	13.90	-0.08	

	Mean_R_80	Bright_100	BordLngth_100	Assym_120	GLCM3_120	ShpIndx_140	\
0	237.23	227.19	56	0.70	3806.36	1.42	
1	217.71	205.53	1526	0.81	1450.14	4.97	
2	234.48	222.97	566	0.81	1094.04	2.08	
3	245.53	232.09	1464	0.92	1125.38	2.88	
4	203.81	196.33	2474	1.00	1146.38	12.06	

	SD_NIR_140
0	26.18
1	22.29
2	15.59
3	13.51
4	15.65

```
In [33]: X_test_final = pd.DataFrame(X_test.iloc[:,feat_cols])

In [34]: X_test_final.head()
```

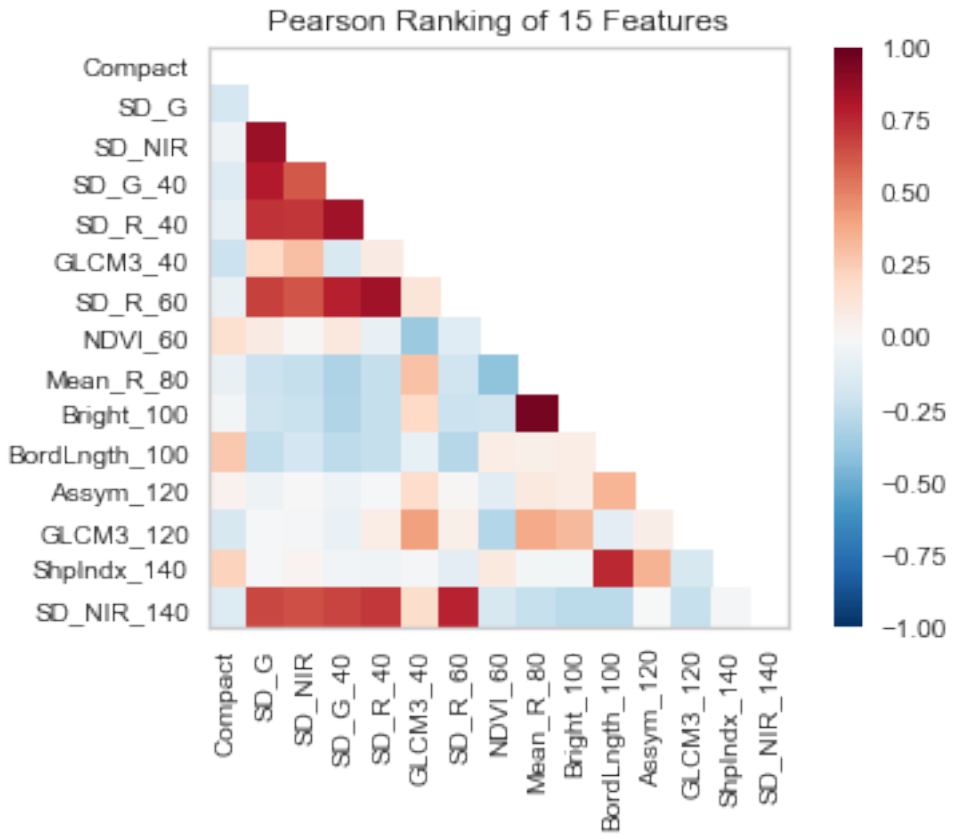
Out[34]:

	Compact	SD_G	SD_NIR	SD_G_40	SD_R_40	GLCM3_40	SD_R_60	NDVI_60	\
0	1.66	11.24	11.24	11.24	11.47	4322.47	20.90	-0.10	
1	1.41	8.15	8.36	8.84	9.97	3063.33	11.89	-0.14	

```
2     1.37   8.11    9.61    10.54    10.81   2668.96    10.70    -0.08
3     3.41  28.60   15.09    42.07    27.05   1234.84    27.05     0.23
4     3.06   5.41    5.27    15.39    16.50   2574.50    23.28    -0.09

   Mean_R_80  Bright_100  BordLngth_100  Assym_120  GLCM3_120  ShpIndx_140  \
0     192.34      181.04            576       0.96    1298.99         3.95
1      51.21       49.82            196       0.86    2659.74         1.32
2      46.34       46.57            602       0.93    1432.44         3.89
3      72.76       69.80            524       0.96     891.36         3.22
4     114.11       96.24            496       0.08    1194.76         2.56

   SD_NIR_140
0       31.15
1       12.01
2       18.75
3       27.67
4       32.05
```

The Top 15 selected features and the co-relation between them are below:

```
In [37]: print(cols)

Index(['Compact', 'SD_G', 'SD_NIR', 'SD_G_40', 'SD_R_40', 'GLCM3_40',
       'SD_R_60', 'NDVI_60', 'Mean_R_80', 'Bright_100', 'BordLngth_100',
       'Assym_120', 'GLCM3_120', 'ShpIndx_140', 'SD_NIR_140'],
      dtype='object')


In [38]: pearson_visual = Rank2D(features = cols, algorithm = "pearson")
         pearson_visual.fit(X_train_final, Y_train)
         pearson_visual.transform(X_train_final)
         pearson_visual.poof()
```



Feature Selection with 10 fold Cross validation gave better accuracy scores compared to CV with 5 folds. Hence we decided to use Feature selection with 10 Cross Validation folds

3.5 2. Using PCA for Components Generation

Principal Component Analysis (PCA) facilitates dimensionality reduction.We will now implement PCA on our entire dataset and not the subsetted dataset, for generating the principal components for our model.

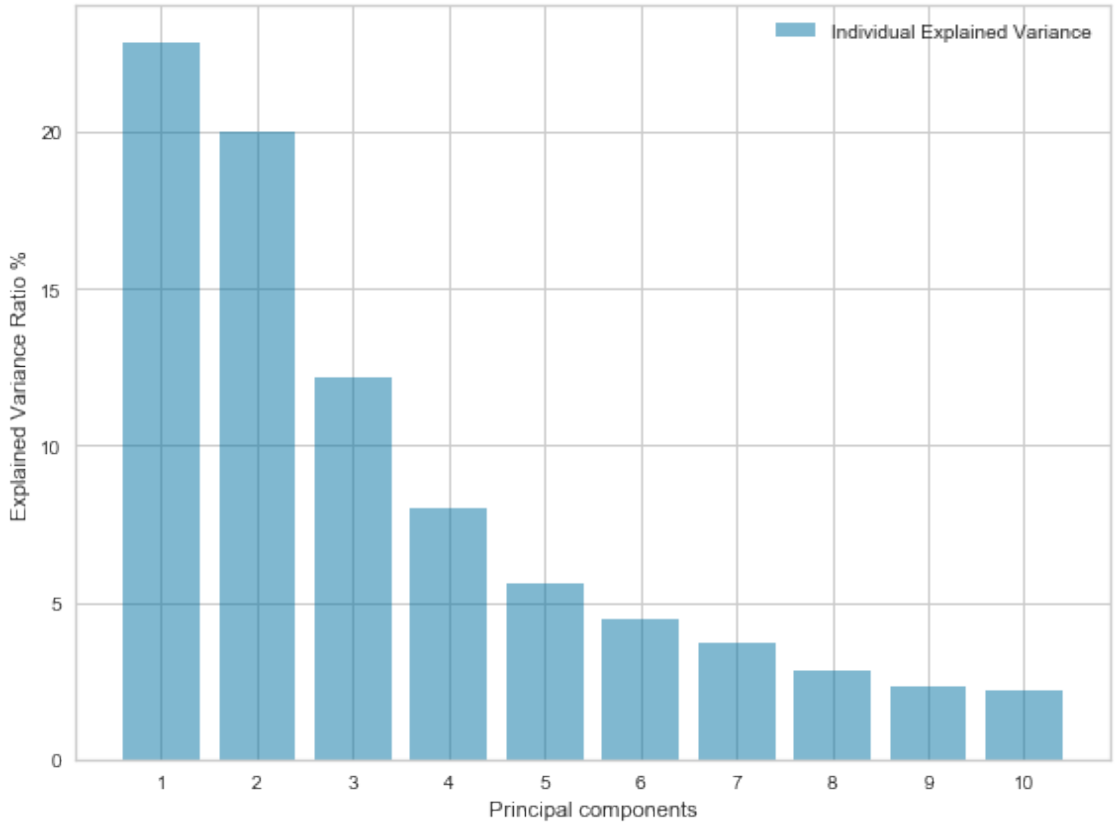
This was not related to feature Selection and its just and indeendent experiment to see how many componenets would be good enough to get maximum explained variance. We havent used PCA for classfication in our model.

```
In [43]: from sklearn.decomposition import PCA
         from sklearn.preprocessing import StandardScaler
         # Standardizing the features
         X_t = StandardScaler().fit_transform(X_train)
         pca = PCA(n_components = 10)
         X = pca.fit_transform(X_t)
         explained_variance = pca.explained_variance_ratio_
         print(explained_variance*100 )
         components = list(range(1,11))
```

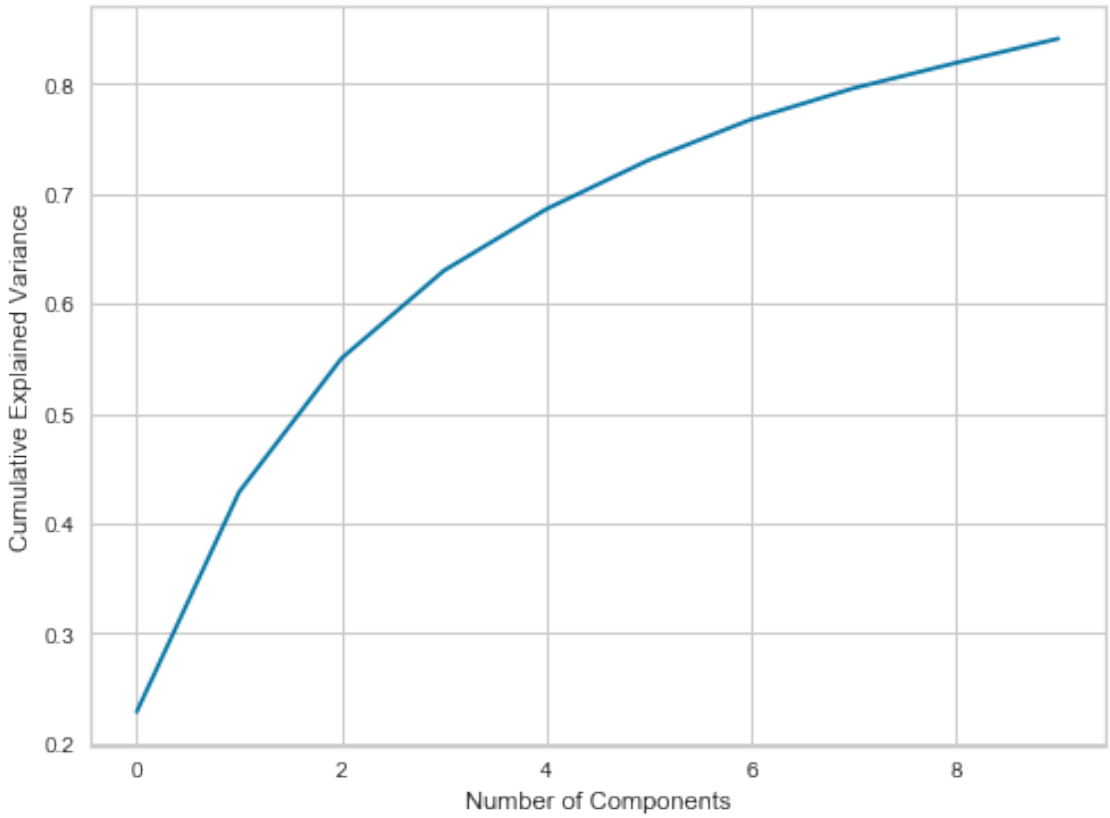
```
[22.87881222 19.99515707 12.16194706  7.98098367  5.59915444  4.46740385
  3.69810475  2.83170874  2.31035923  2.20652465]
```

Based on the above output we will select the first **10** components for our model. Below we plot the graph for the amount of variance explained by each component.

```
In [44]: plt.figure(figsize=(8,6))
         plt.bar(components, explained_variance*100, alpha=0.5, align='center',tick_label = [1
                 label='Individual Explained Variance')
         plt.ylabel('Explained Variance Ratio %')
         plt.xlabel('Principal components')
         plt.legend(loc='best')
         plt.tight_layout()
         plt.show()
```




```
In [45]: plt.figure(figsize=(8,6))
         plt.plot(np.cumsum(pca.explained_variance_ratio_))
         plt.xlabel('Number of Components')
         plt.ylabel('Cumulative Explained Variance');
```

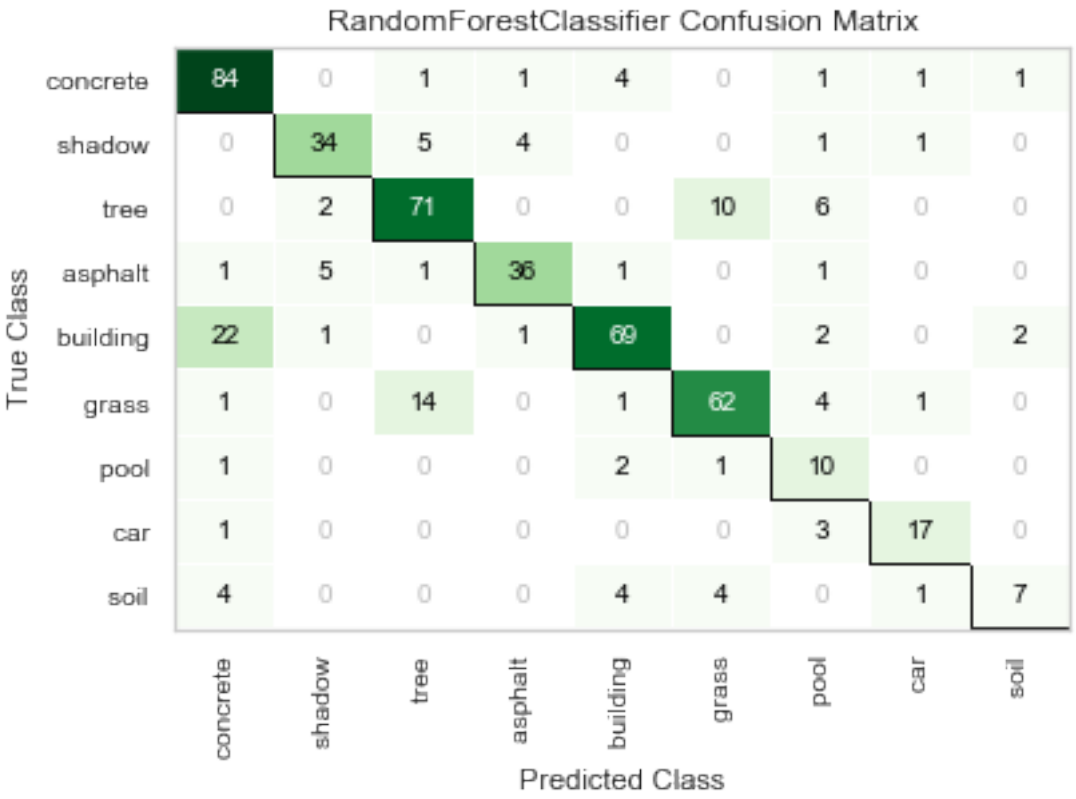


4 Model Selection

4.1 Classification using the Random Forest Classifier

```
In [48]: rf_classifier = RandomForestClassifier(random_state=42)
         rf_cm = ConfusionMatrix(rf_classifier, classes = list(Y_test.unique()), cmap = "Greens
         rf_cm.fit(X_train_final,Y_train)
         rf_cm.score(X_test_final,Y_test)
         y_pred_rf = rf_cm.predict(X_test_final)
         accuracy_rf = accuracy_score(Y_test, y_pred_rf , sample_weight=None)
         print('Random Forest' ,accuracy_rf)
         rf_cm.poof()
```

```
Random Forest 0.7692307692307693
```



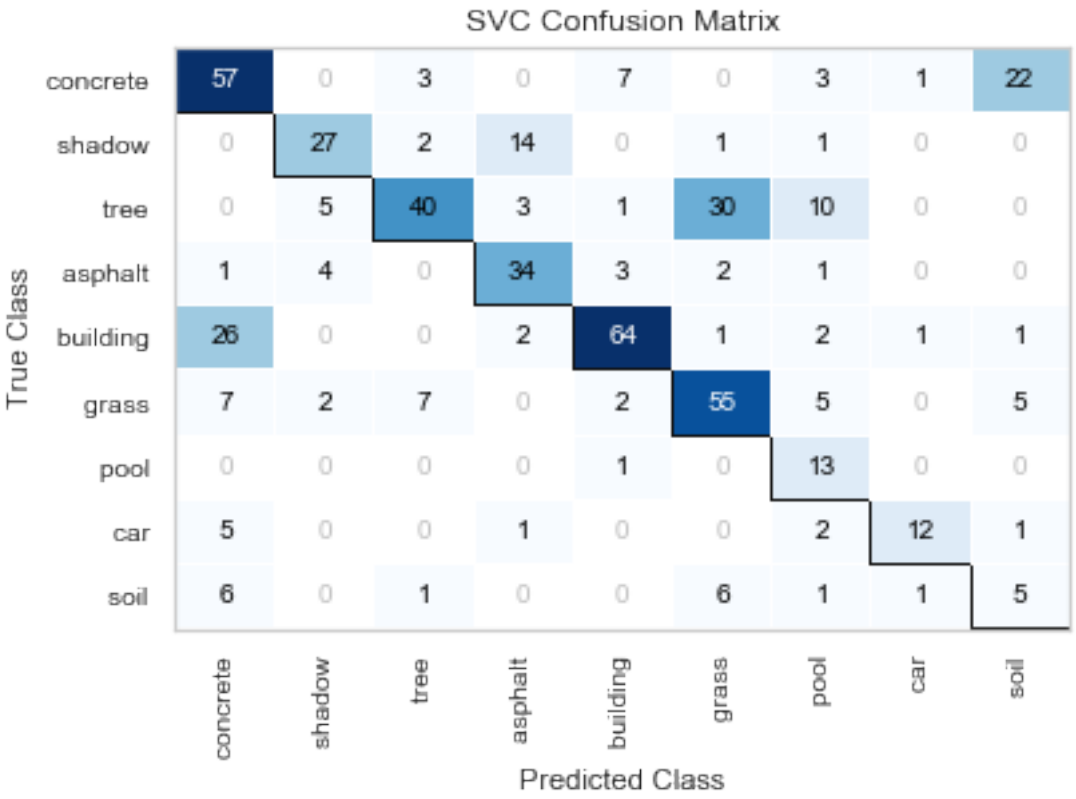
4.2 Classification using Linear SVM

Support Vector Machines (SVM's) work best with data associated with images. This was the intuition behind utilising SVM. The kernels that were experimented with include - linear and poly .

- Linear SVM is a parametric model, an RBF kernel SVM isn't, and the complexity of the latter grows with the size of the training set.

```
In [49]: svml_classifier = svm.SVC(kernel = "linear", random_state=50)
         svml_cm = ConfusionMatrix(svml_classifier, classes = list(Y_test.unique()), cmap = "Bl
         svml_cm.fit(X_train_final,Y_train)
         svml_cm.score(X_test_final,Y_test)
         y_pred_svml = svml_cm.predict(X_test_final)
         accuracy_svml = accuracy_score(Y_test, y_pred_svml , sample_weight=None)
         print('Linear' ,accuracy_svml)
         svml_cm.poof()
```

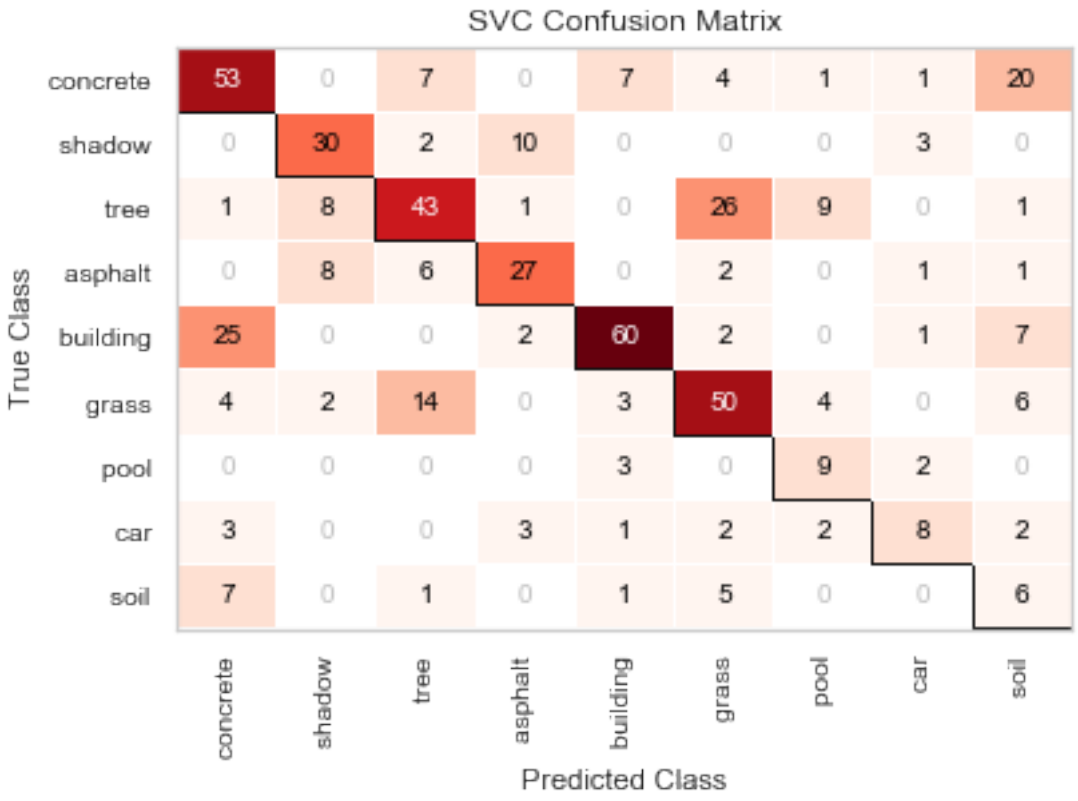
```
Linear 0.6055226824457594
```



4.3 Classification using Polynomial SVM

```
In [50]: svmp_classifier = svm.SVC(kernel = "poly", random_state=50)
         svmp_cm = ConfusionMatrix(svmp_classifier, classes = list(Y_test.unique()), cmap = "R
         svmp_cm.fit(X_train_final,Y_train)
         svmp_cm.score(X_test_final,Y_test)
         y_pred_svmp = svmp_cm.predict(X_test_final)
         accuracy_svmp = accuracy_score(Y_test, y_pred_svmp , sample_weight=None)
         print('Poly' ,accuracy_svmp)
         svmp_cm.poof()
```

```
Poly 0.5641025641025641
```



5 Model Evaluation

5.1 Confusion Matrix

A confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class. This is the key to the confusion matrix.

Procedure for Confusion Matrix calculation:

1. You need a test dataset or a validation dataset with expected outcome values.
2. Make a prediction for each row in your test dataset.
3. From the expected outcomes and predictions count: (a)The number of correct predictions for each class. (b)The number of incorrect predictions for each class, organized by the class that was predicted.

Evaluation Metrics used - Precision, Recall and F-measure are the most commonly used evaluation metrics for classification problems. To get the value of precision we divide the total number of correctly classified positive examples by the total number of predicted positive examples. High Precision indicates an example labeled as positive is indeed positive.

Precision is defined as the number of true positives divided by the number of true positives plus the number of false positives. It is the ability of a classification model to identify only the relevant data points.

5.2 Q.2 How did you choose which performance metrics to report?

The primary output we obtained after the classification task was a confusion matrix. Since a confusion matrix gives an indication of the number of true and false positives, the metrics that seemed to be most relevant are Precision and Recall.

5.3 Precision score for RF Classifier

```
In [51]: from sklearn.metrics import precision_score
         precision_score_rf = precision_score(Y_test, y_pred_rf,average='macro')
         print(precision_score_rf)
```

```
0.7443290251197701
```

5.4 Precision score for Linear SVM Classifier

```
In [52]: from sklearn.metrics import precision_score
         precision_score = precision_score(Y_test, y_pred_svml,average='macro')
         print(precision_score)
```

```
0.5935911923982359
```

5.5 Precision score for Polynomial SVM Classifier

```
In [53]: from sklearn.metrics import precision_score
         precision_score = precision_score(Y_test, y_pred_svmp,average='macro')
         print(precision_score)
```

```
0.5289806643249285
```

- Recall

Recall is the ability of a model to find all the relevant cases within a dataset. It is the number of true positives divided by the number of true positives plus the number of false negatives. True positives are data point classified as positive by the model that actually are positive (meaning they are correct), and false negatives are data points the model identifies as negative that actually are positive (incorrect).

5.6 Recall score for RF Classifier

```
In [54]: recall_score(Y_test, y_pred_rf, average="macro")
```

```
Out[54]: 0.7320746503109169
```

5.7 Recall score for Linear SVM Classifier

```
In [55]: recall_score(Y_test, y_pred_svml, average="macro")
```

```
Out[55]: 0.6100379333835363
```

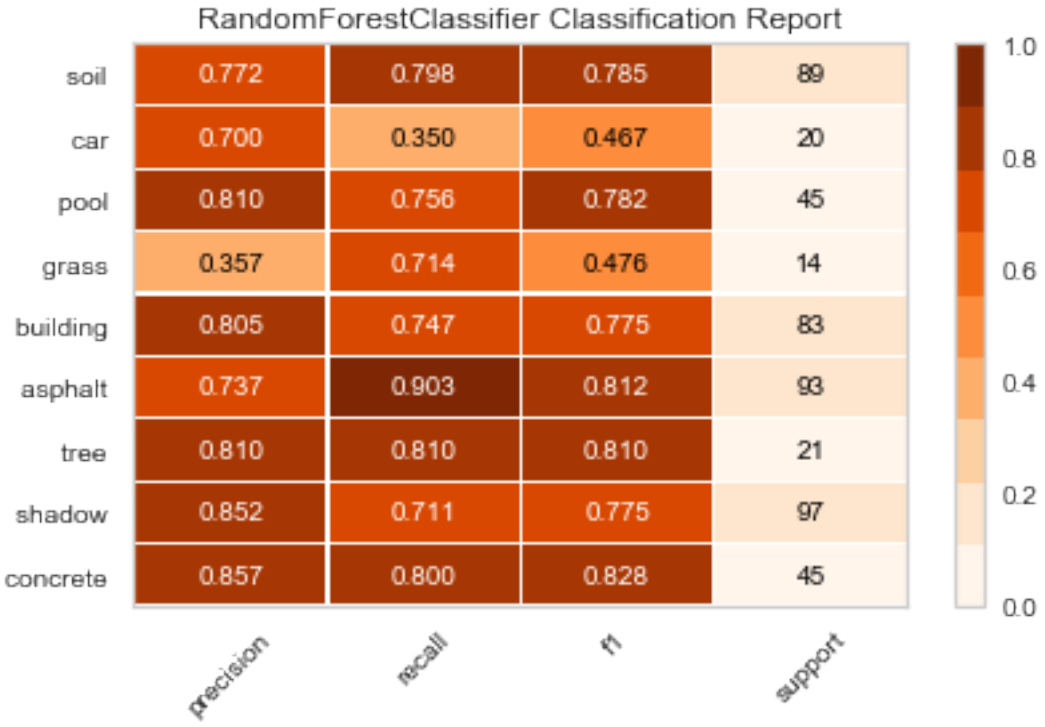
5.8 Recall score for Polynomial SVM Classifier

```
In [56]: recall_score(Y_test, y_pred_svmp, average="macro")
```

```
Out[56]: 0.5404978967328161
```

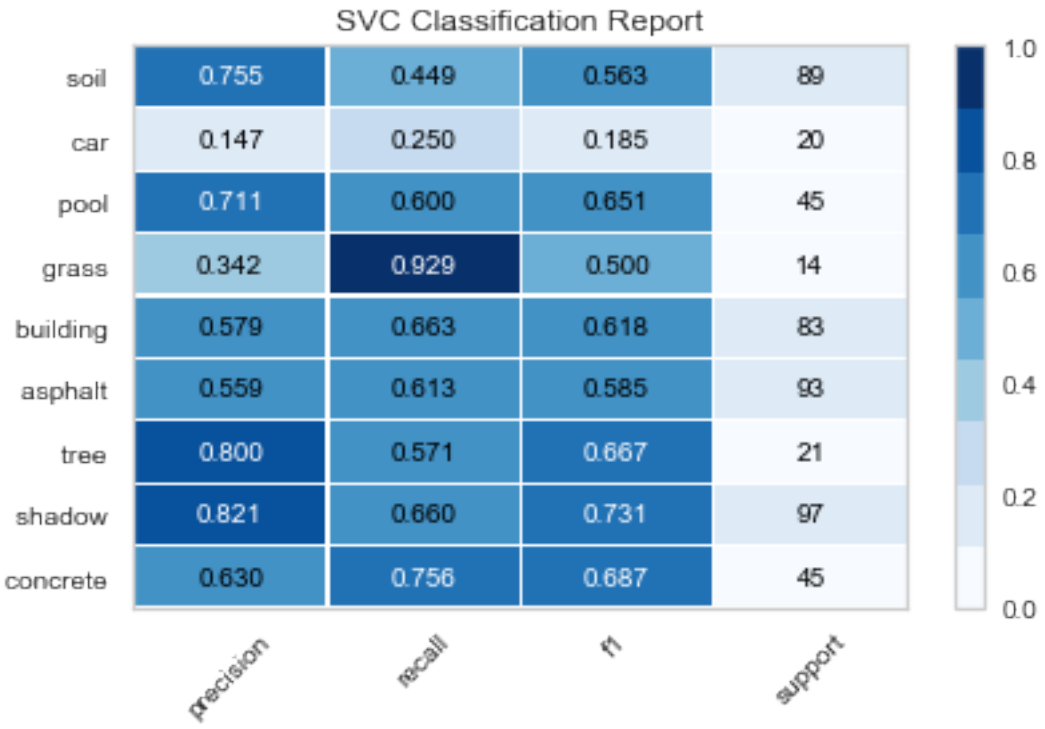
5.9 Classification Report for RF Classifier

```
In [57]: rf_visual = ClassificationReport(rf_classifier, classes = list(Y_test.unique()), cmap
         rf_visual.fit(X_train_final,Y_train)
         rf_visual.score(X_test_final,Y_test)
         g = rf_visual.poof()
```



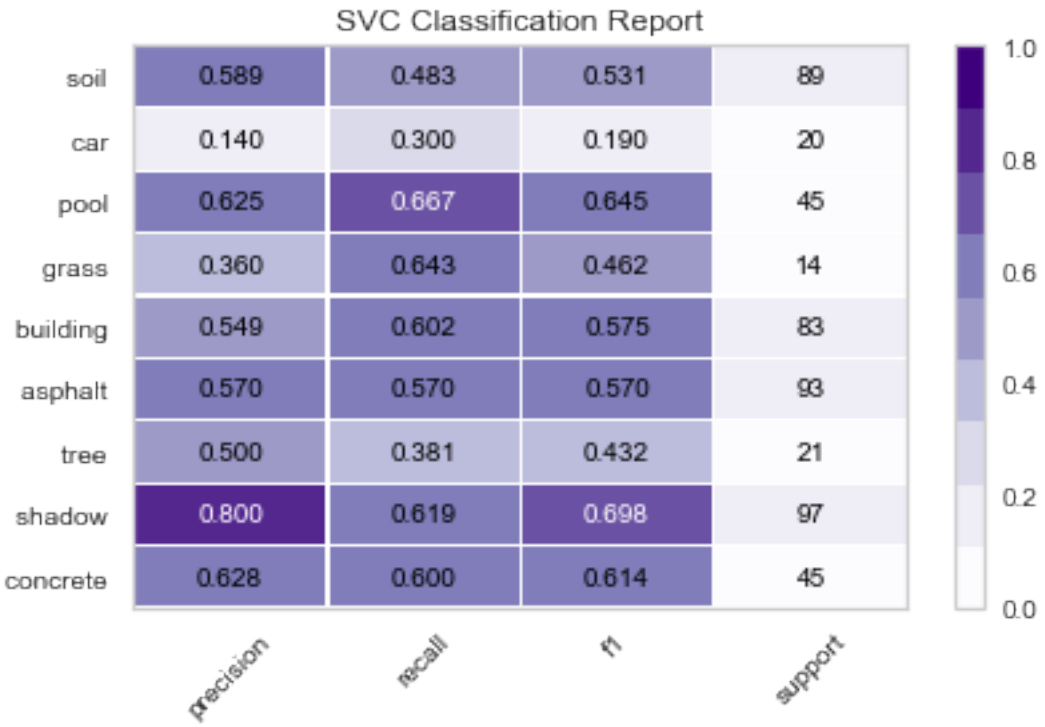
5.10 Classification Report for Linear SVM Classifier

```
In [58]: svml_visual = ClassificationReport(svm.SVC(kernel = "linear", random_state=50), classe
         svml_visual.fit(X_train_final,Y_train)
         svml_visual.score(X_test_final,Y_test)
         g = svml_visual.poof()
```



5.11 Classification Report for Polynomial SVM Classifier

```
In [59]: svmp_visual = ClassificationReport(svm.SVC(kernel = "poly", random_state=50), classes
         svmp_visual.fit(X_train_final,Y_train)
         svmp_visual.score(X_test_final,Y_test)
         g = svmp_visual.poof()
```



- Cohen's Kappa Score

Kappa measures the percentage of data values in the main diagonal of the table and then adjusts these values for the amount of agreement that could be expected due to chance alone.

5.12 Cohen's Kappa score for Random Forest Classifier

```
In [63]: cohen_kappa_score(Y_test, y_pred_rf)
```

```
Out[63]: 0.729381654942107
```

5.13 Cohen's Kappa score for Linear SVM Classifier

```
In [64]: cohen_kappa_score(Y_test, y_pred_svml)
```

```
Out[64]: 0.5426829268292683
```

5.14 Cohen's Kappa score for Polynomial SVM Classifier

```
In [65]: cohen_kappa_score(Y_test, y_pred_svmp)
```

```
Out[65]: 0.49399369563842954
```

5.15 Q.3 Are there any issues with your model that are apparent from the confusion matrix, but not from your performance metrics?

No, The results obtained from the confusion matrix seem to be consistent with what we see in the performance metrics.

5.16 References

Abe2010_Chapter_FeatureSelectionAndExtraction.pdf | TowardsDataScience | kdNuggets