A2 GROUP 12 : SHRUTI SANKOLLI, PAULAMI RAY, SAUMIL SHAH

Assignment 2: Classification

Classification Task:

Q. How did you choose which classification algorithm would be appropriate for these data?

Solution: We observed a number of things to decide which classification algorithm to use for this data. We know that this is a multiclass problem with 9 target labels and all classification algorithms are not good with handling multiclass problems. Thus, initially we thought of using a Tree based algorithm which can work well with such problems. We also noticed that the data has a very high number of features (148) and thus it would be crucial to know how important these features are in the classification of target labels. Based on these initial reasons we decided to use Random Forest algorithm for classification. Random forest is a classifier which works well with multiclass problems and also allows us to see what features or variables are contributing to the classification and their relative importance based on their location depthwise in the tree. We also know that for this problem the training data is too small as compared to the test data and has only 168 samples. It is difficult for general classification algorithms to learn from a low training sample of data which could often lead to problems like overfitting. Random Forest is a classifier which is quite robust and does not get affected to such problems. Generating “n” number of trees and combining the overall predictions of all the trees can also help in reducing the variance and thus improving the overall classification rate. Being a non-parametric algorithm it also does not make any assumptions about the training or the unseen testing data and thus gives a better performance. Random Forest also allows us to tune parameters like “n_estimators” and “max_features” which can again help in improving the results.

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

Solution: We chose the following metrics to report:

1. **Accuracy**: Classification accuracy was our starting point and we calculated it with the number of correct predictions divided by the total number of predictions.
2. **Precision:** Precision was used to measure the classifier exactness of our model.We calculated it with the number of positive predictions divided by the total number of positive class values predicted.
3. **Recall**: We used Recall to check the classifier completeness of our model.It is calculated as the number of positive predictions divided by the number of positive class values in the test data.

4. **F1-score**: We used F1 score as it measures the balance between the precision and recall.We calculated it as 2*((precision*recall)/(precision+recall)) in our model.

Usually in classification tasks, just considering accuracy is not right. Because accuracy is a very biased interpretation of the model. To actually understand the model performance for a classification task it is really important to consider other parameters like precision and recall which give a clear understanding of how the model is performing on the test data. F1 score gives a general interpretation of both precision and recall together. Apart from these parameters we also considered confusion matrix because it gives us a better understanding of the classification and misclassification rates for every single class which is important for model performance analysis.

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

Solution: From the performance metrics, we could determine whether the model is performing well or not. But since the Confusion Matrix provides the prediction results in a clean and unambiguous way we got the specific details on the misclassifications using it. From all three confusion matrices for different threshold values (Fig 10, Fig 14 and Fig 16) we can clearly see that shadow is getting misclassified as asphalt and soil is getting misclassified as building. This difference is not clearly apparent in the performance parameters. Thus, because of the confusion matrix we are clearly able to understand that our model is not able to correctly differentiate between shadow & asphalt and soil & building. We also noticed a decrease in this misclassification rate as we keep increasing the number of features. This also indicates that each and every feature does contain some important information and by not retaining them we do lose that information.

Description of the Classification Process used in our model:

- **Exploratory Analysis:**

  Training Data:
  1) Has 168 rows and 148 columns.

  Test Data:
  1) Has 507 rows and 148 columns.

  Preliminary Check for missing, null and incorrect values:
  2) No nulls or NaN's found in the training or test data. All data values are present.

Distribution of the target “class” variable can be seen in Fig 1.

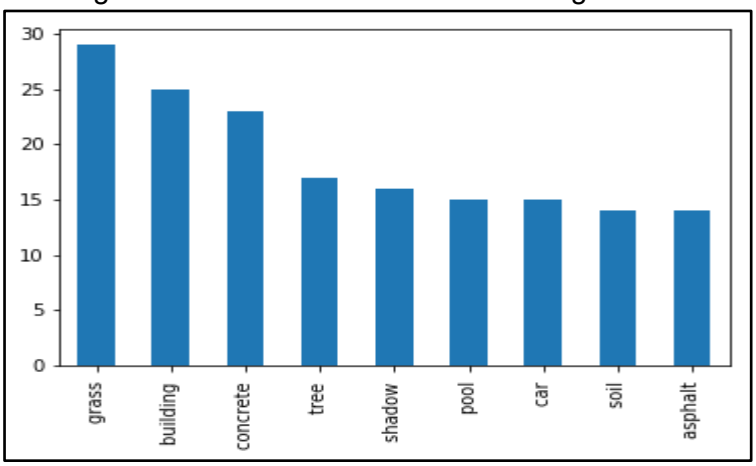

Fig 1: Class distribution for the training set

Data in the training set is well distributed among the different classes, hence there was no need of further preprocessing

- **Classification Tasks (Feature Selection and Hyperparameter Tuning):**

We followed the steps below as part of the classification task:

1) **Read Input:** We read the training data set into a pandas dataframe.

2) **Select the predictors and target:** We selected the class variable as the target. Remaining 147 features formed the predictors. Fig 2 shows the code snippet for the predictors and target.

```
predictors=train_data.drop(["class"],axis=1)
target=train_data["class"]
```

Fig 2: Predictors and target - Training set

3) **Base Model:** We fitted the base model (Random Forest) with no hyperparameter tuning and no feature selection. We estimated the accuracy of this model with n estimators set to 10.
Accuracy came to 77%.
Fig 3 shows the code snippet for the base model and its accuracy.

```
# Base model with no hyperparameter tuning
from sklearn import metrics
base_model = RandomForestClassifier(n_estimators = 10, random_state = 42)
base_model.fit(predictors, target)
predictions = base_model.predict(predictors_test)
print("Accuracy is:",metrics.accuracy_score(target_test,predictions))
```

```
Accuracy is: 0.777120315582
```

Fig 3: Accuracy of the base model

4) **a) Feature Selection based on the base model:** We performed feature selection based on the feature importances returned by the base model.Fig 4. shows the feature importances for some predictors as returned by the base model.

```
('BrdIndx', 0.0034763253070681926)
('Area', 0.022952227924512212)
('Round', 0.00068588225687923584)
('Bright', 0.0033281674902774921)
('Compact', 0.00243981766614593087)
('ShpIndx', 0.0012977554128318883)
('Mean_G', 0.0072705552785245434)
('Mean_R', 0.01165299713109697)
('Mean_NIR', 0.013640597070248464)
('SD_G', 0.0)
('SD_R', 0.0089419580597286073)
('SD_NIR', 0.0093915144981367594)
('LW', 0.0034702813078158075)
('GLCM1', 0.0023327718079373888)
('Rect', 0.0055648286281536202)
('GLCM2', 0.00028986290024935363)
('Dens', 0.0)
('Assym', 0.0)
('NDVI', 0.068453899977002269)
```

Fig 4: Feature importance based on the base model

b) Hyperparameter Tuning: We used RandomizedSearchCV available from the python scikit library to perform hyperparameter tuning for our RandomForestClassifier.

To do so, we provided a range of values for the different hyperparameters and let RandomizedSearchCV try all combinations of the hyperparameters values on the training data and return the best set of hyperparameter values to use.Fig 5 shows the default hyperparameter values for the RandomForestClassifier.

```
#Default hyperparameters for random forest
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(random_state = 42)
from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(rf.get_params())
```

```
Parameters currently in use:

{'bootstrap': True,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 '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,
 'n_estimators': 10,
 'n_jobs': 1,
 'oob_score': False,
 'random_state': 42,
 'verbose': 0,
 'warm_start': False}
```

Fig 5: Default hyperparameter values

Fig 6. Shows the range of values for the different hyperparameters.

```
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)
```

```
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
```

Fig 6: Range of values for the hyperparameter tuning

5) **Classification:**

a) **Feature Selection at 0.005 threshold:**

Based on the feature importances returned in the previous step, we arbitrarily decided a threshold value of 0.005. So any feature value above the threshold were considered important and the rest were discarded. Fig 7 shows the feature selection at 0.005 threshold.

```
from sklearn.feature_selection import SelectFromModel
sfm = SelectFromModel(base_model, threshold=0.005)

# Train the selector
sfm.fit(predictors, target)
```

```
SelectFromModel(estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', 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, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=42, verbose=0, warm_start=False),
       norm_order=1, prefit=False, threshold=0.005)
```

Fig 7: Feature selection at 0.005

This threshold value retained only **58 features out of the total 147 features.**

```
a=[]
for feature_list_index in sfm.get_support(indices=True):
    a.append(feature_list_index)

print(len(a))
```

```
58
```

Fig 8: No of features retained at 0.005 threshold

- **Hyperparameter Tuning:** We then performed hyperparameter tuning at 0.005 threshold to get the best combinations of hyperparameters. Fig 9 shows the best set of hyperparameters for the training set at 0.005 threshold is:

```
{'bootstrap': False,
 'max_depth': 10,
 'max_features': 'sqrt',
 'min_samples_leaf': 2,
 'min_samples_split': 5,
 'n_estimators': 1200}
```

Fig 9: Best hyperparameter combination at 0.005 threshold

- **Classification on the test set:** We then apply the trained model with 58 features as returned by fig 8 and with the hyperparameters shown in fig 9. to the test set. The performance metrics and the confusion matrix can be seen in Fig 10.

  Metrics: Accuracy: 79%

	precision	recall	f1-score	support
concrete	0.92	0.78	0.84	45
shadow	0.90	0.64	0.75	97
tree	0.65	0.95	0.77	21
asphalt	0.75	0.88	0.81	93
building	0.75	0.88	0.81	83
grass	1.00	0.93	0.96	14
pool	0.77	0.91	0.84	45
car	0.56	0.70	0.62	20
soil	0.87	0.70	0.78	89
avg / total	0.81	0.79	0.79	507

Confusion Matrix:

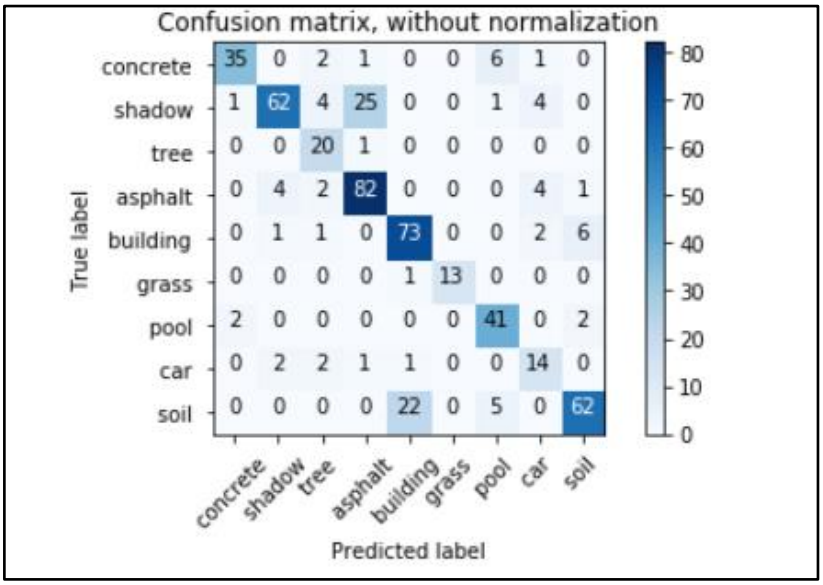

Fig 10: Precision Recall and Confusion Matrix

From the confusion matrix we can see that high number of instances belonging to class shadow have been misclassified as asphalt. Our hypothesis is that the misclassification is mostly because of similarity in the appearance of an asphalt surface and a shadow.

b) **Feature Selection at 0.01 threshold:**

We tightened the feature selection on our model to assess model sensitivity to the predictors. So we reduced the threshold value to 0.01 from 0.005. So any feature value above the threshold were considered important and the rest were discarded. Fig 11 shows the feature selection at 0.01 threshold.

```
from sklearn.feature_selection import SelectFromModel
sfm = SelectFromModel(base_model, threshold=0.01)

# Train the selector
sfm.fit(predictors, target)
```

```
SelectFromModel(estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', 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, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=42, verbose=0, warm_start=False),
        norm_order=1, prefit=False, threshold=0.01)
```

Fig 11: Feature selection at 0.01

This threshold value retained only **35 features out of the total 147 features.**

```
a=[]
for feature_list_index in sfm.get_support(indices=True):
    a.append(feature_list_index)

print(len(a))
```

```
35
```

Fig 12: No of features retained at 0.01 threshold

- **Hyperparameter Tuning :**We then perform hyperparameter tuning at 0.01 threshold to get the best combinations of hyperparameters.
  Fig 9 shows the best set of hyperparameters for the training set at 0.01 threshold is:

```
{'bootstrap': False,
 'max_depth': 70,
 'max_features': 'sqrt',
 'min_samples_leaf': 1,
 'min_samples_split': 5,
 'n_estimators': 1600}
```

Fig 13: Best hyperparameter combination at 0.01 threshold

- **Classification on the test set:** We then apply the trained model with 35 features returned by fig 12 and with the hyperparameters shown in Fig 13. to the test set. The performance metrics and the confusion matrix can be seen in Fig 14.
  Metrics: Accuracy reduced to 78%

	precision	recall	f1-score	support
concrete	0.89	0.76	0.82	45
shadow	0.89	0.67	0.76	97
tree	0.59	0.95	0.73	21
asphalt	0.77	0.87	0.82	93
building	0.73	0.89	0.80	83
grass	1.00	0.93	0.96	14
pool	0.75	0.89	0.82	45
car	0.54	0.65	0.59	20
soil	0.89	0.66	0.76	89
avg / total	0.81	0.79	0.79	507

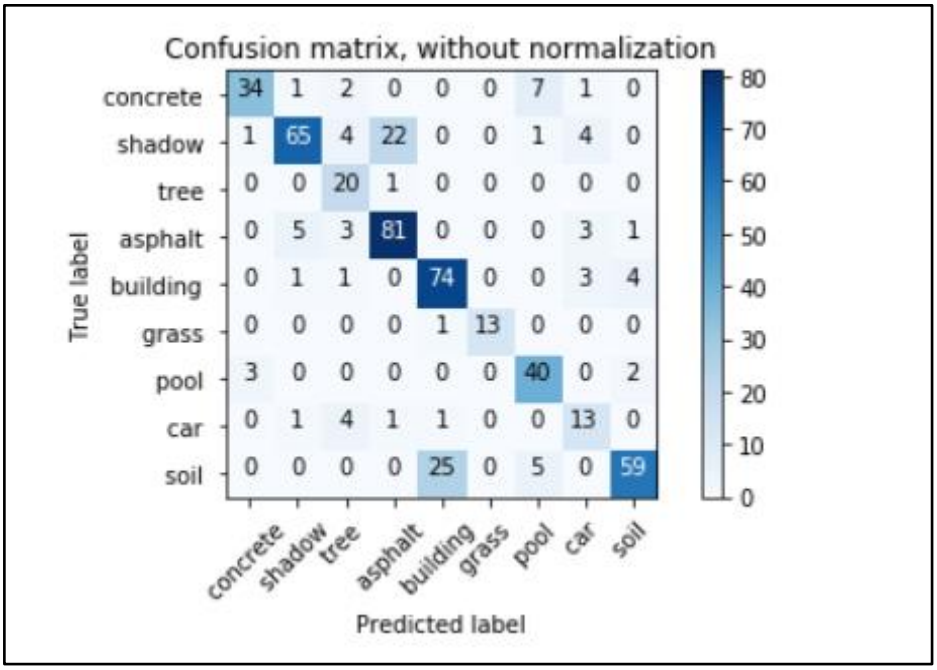

Fig 14: Precision Recall and Confusion Matrix

From the confusion matrix, we can observe that the large number of instances belonging to class soil have been misclassified as building. Our hypothesis is that some distinguishing features(with respect to soil and building) were dropped from the final list due to the threshold value, thereby increasing the number of misclassifications.

c) **Feature Selection with no threshold (Retain all 147 features):**
We did not discard any feature in this model to assess whether model performance improves with more features and also to determine whether each feature is relevant to the model.

- **Hyperparameter Tuning:**We then perform hyperparameter tuning with all features to get the best combinations of hyperparameters.
Fig 15 shows the best set of hyperparameters for the training set at full capacity:

```
{'bootstrap': True,
 'max_depth': 80,
 'max_features': 'auto',
 'min_samples_leaf': 1,
 'min_samples_split': 5,
 'n_estimators': 600}
```

Fig 15: Best hyperparameter combination at 0.01 threshold

- **Classification on the test set:** We then apply the trained model with all features and with the hyperparameters shown in Fig 15. to the test data set. Fig 16. Shows the performance metrics and the confusion matrix.

  Metrics:
  - Accuracy is highest at 81%

	precision	recall	f1-score	support
concrete	0.95	0.82	0.88	45
shadow	0.88	0.72	0.79	97
tree	0.67	0.95	0.78	21
asphalt	0.79	0.89	0.84	93
building	0.76	0.88	0.82	83
grass	1.00	0.86	0.92	14
pool	0.79	0.93	0.86	45
car	0.58	0.55	0.56	20
soil	0.88	0.72	0.79	89
avg / total	0.82	0.81	0.81	507

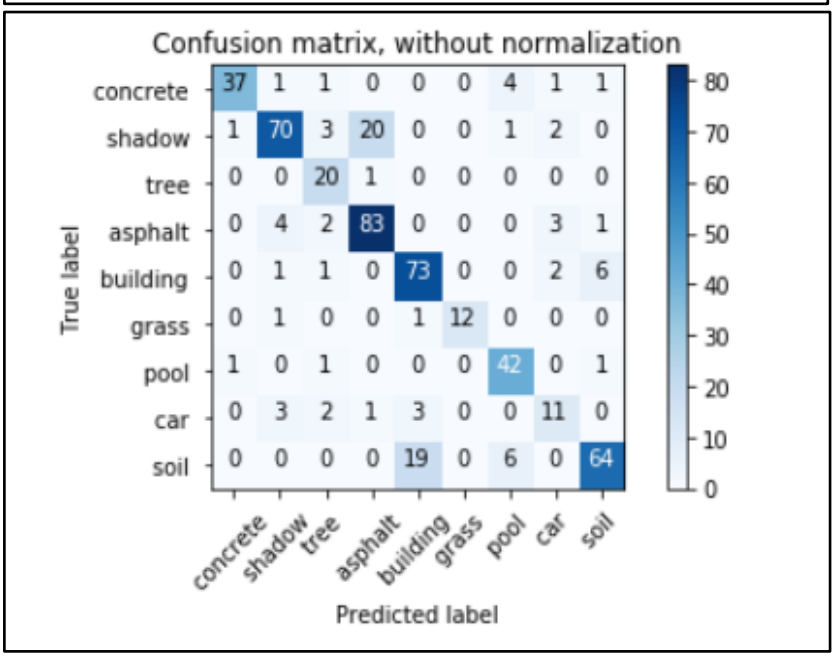

Fig 16: Precision Recall and Confusion Matrix

The confusion matrix now shows high numbers for correct classifications.

Based on the model outputs at 0.005 , 0.01 threshold and no threshold, we concluded the following preliminary results:

- Model had lowest accuracy and bad metrics at the highest threshold value i.e 0.01. As the threshold was relaxed, the model performed better.
- Most of the variables are somewhat relevant to the model; reducing the variables has a direct effect on the accuracy.
- Reducing the threshold below 0.005 did not yield significant change in the number of features retained.
- **For a small cost in accuracy, we can achieve significant reduction in dimensionality.**

6. Feature ranking: In order to get a better understanding of the impact of predictors on the target, we plotted the feature ranking of the no threshold model (as it has the best results so far).

```
Feature ranking:
1. feature 18 (0.046050)
2. feature 39 (0.035563)
3. feature 60 (0.032472)
4. feature 81 (0.028617)
5. feature 6 (0.020592)
6. feature 144 (0.018461)
7. feature 66 (0.018011)
8. feature 87 (0.017995)
9. feature 29 (0.017954)
10. feature 102 (0.017137)
11. feature 123 (0.016762)
12. feature 3 (0.016473)
13. feature 7 (0.016301)
14. feature 48 (0.015826)
15. feature 70 (0.015683)
16. feature 71 (0.015103)
17. feature 28 (0.014835)
18. feature 27 (0.014464)
19. feature 8 (0.014237)
20. feature 92 (0.013698)
```

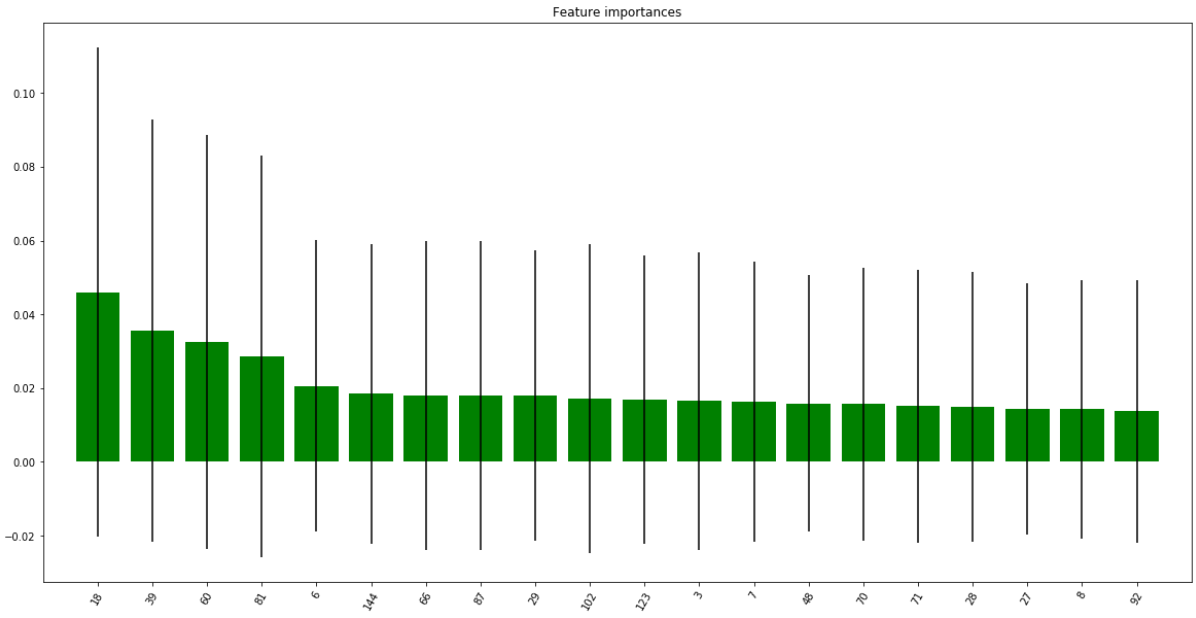

Fig 17: Top 20 features based on Feature Importance

Bonus Task:

Alternative Approach for Classification - SVC with RFE (Recursive Feature Elimination)

Support Vector Classifiers are another family of classifiers that can be used for multiclass classification.Since the classification task in this assignment is multiclass, it is worthwhile to check the performance of SVC on the Urban Land cover dataset.
This also provided us with a basis to compare our Random Forest Classification results.

- SVC with RFE : In this classification approach, we are using Stratified 10 fold feature elimination to fit a SVC model to the data.
- The RFECV uses an “accuracy” scoring model to determine the optimal number of features needed and then trains the model based on those features.

```
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from sklearn.feature_selection import RFECV

# Create the RFE object and compute a cross-validated score.
svc = SVC(kernel="linear")
# The "accuracy" scoring is proportional to the number of correct
# classifications
rfecv = RFECV(estimator=svc, step=1, cv=10,
              scoring='accuracy')
rfecv.fit(predictors, target)

print("Optimal number of features : %d" % rfecv.n_features_)

# Plot number of features VS. cross-validation scores
plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (nb of correct classifications)")
plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_)
plt.show()
```

```
Optimal number of features : 20
```

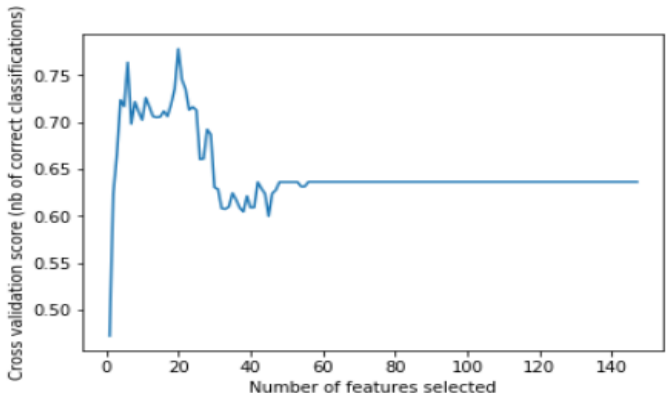

Fig 18: Number of features selected with SVC - RFECV

For the urban land cover data, the RFECV model returns 20 as the optimal number of features.

Fig shows the cross validation score for different number of features. The score is highest at 20 and drops until 60. After 60, the score remains stagnant.

The 20 features retained by this model are:

```
Chosen best feature by rfe with 10-fold cross-validation is : Index(['Bright', 'Mean_G', 'Mean_R', 'Mean_NIR', 'Mean_G_40', 'Mean_R_40',
       'Mean_NIR_40', 'Mean_G_60', 'Mean_R_60', 'Mean_NIR_60', 'Mean_R_80',
       'Mean_NIR_80', 'Mean_G_100', 'Mean_R_100', 'Mean_NIR_100', 'Bright_120',
       'Mean_R_120', 'Bright_140', 'Mean_G_140', 'Mean_R_140'],
      dtype='object')
```

Classification on the test set: Accuracy: 64.87%

Metrics:

Fig 19 shows the performance metrics and the confusion matrix for SVM with RFECV.

	precision	recall	f1-score	support
concrete	0.95	0.82	0.88	45
shadow	0.88	0.72	0.79	97
tree	0.67	0.95	0.78	21
asphalt	0.79	0.89	0.84	93
building	0.76	0.88	0.82	83
grass	1.00	0.86	0.92	14
pool	0.79	0.93	0.86	45
car	0.58	0.55	0.56	20
soil	0.88	0.72	0.79	89
avg / total	0.82	0.81	0.81	507

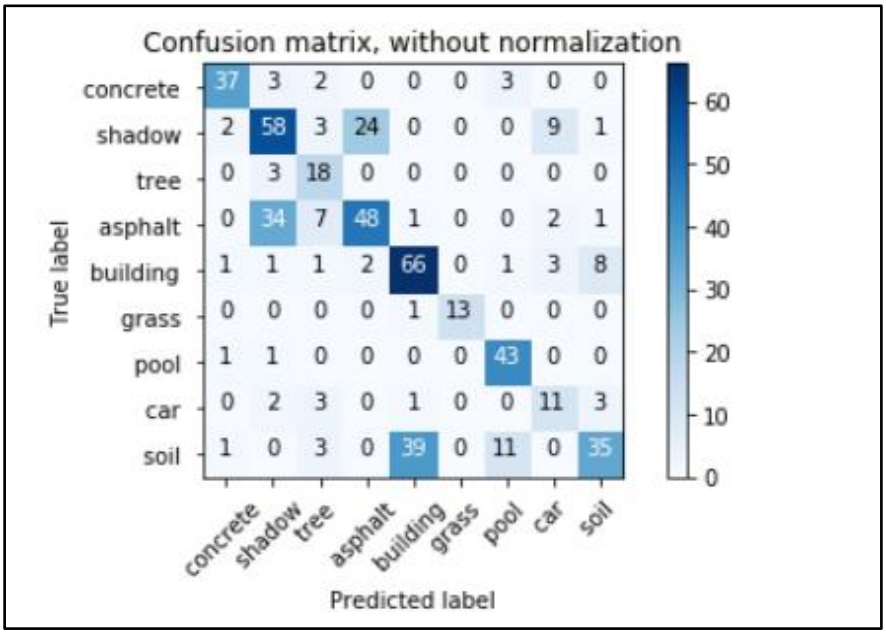

Fig 19: Precision Recall and Confusion Matrix

Conclusion:

Based on the two classifiers that we worked with, we can conclude that the Random Forest classifier, predicted the model well retaining the maximum features in the

data. We also noticed that although feature selection reduces the computational load while running the algorithm, it does compromise on the model performance. We saw this trend of reduction in model performance while testing the Random forest classifier on different number of features (35 features, 58 features and all features). Thus, we conclude that feature selection is something that should be used when it is really important to understand the importance of each individual feature and when we do not want the model to be computationally heavy. In applications where model performance is of more importance than understanding the importance of features we should use a dimensionality reduction approach like Principal Component Analysis. While PCA is not a feature selection method, it does have the capability to retain the importance of each and every feature from the dataset while also reducing the dimensionality (which can help in reducing the computational load). Feature selection completely eliminates some features assuming that they are not important at all. But this is not the case in most applications. Although some features might be less important than others but not considering them at all does affect the model performance. It is thus better to use an approach like PCA which can reduce the number of features into a few components while ensuring to consider the importance of each feature.