In [1]:

```
# Load necessary packages
import numpy as np
import pandas as pd
import matplotlib as mpl

# Visualization modules
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns

# Data-processing modules
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Modeling and evaluation modules
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from sklearn.model_selection import GridSearchCV
from sklearn import metrics

import warnings
warnings.filterwarnings("ignore")

# Module for saving model
import pickle

# Set seed for reproducibility
SEED = 3
```

Part I. Data Processing

a) Import the data: You are provided separate .csv files for train and test.

- Train shape: (507, 148)
- Test shape: (168, 148)

In [2]:

```
# Load data
X_train = pd.read_csv('train_data.csv')
X_test = pd.read_csv('test_data.csv')
# Verify dimension
print(X_train.shape)
print(X_test.shape)
```

```
(507, 148)
(168, 148)
```

b) Remove any rows that have missing data across both sets of data.

In [3]:

```
# Drop rows will null values
X_train.dropna(axis = 0,inplace = True)
X_test.dropna(axis = 0,inplace = True)
# Verify dimension
print(X_train.shape)
print(X_test.shape)
```

```
(507, 148)
(168, 148)
```

c) The target variable (dependent variable) is called "class", make sure to separate this out into a "y_train" and "y_test" and remove from your "X_train" and "X_test".

In [4]:

```
# Create y_train
y_train = X_train['class']
# Drop target variable from X_train
X_train = X_train.drop(['class'], axis=1)
# Repeat the same steps for test data
y_test = X_test['class']
X_test = X_test.drop(['class'], axis=1)
# Verify dimension
print(y_train.shape)
print(X_train.shape)
print(y_test.shape)
print(X_test.shape)
```

```
(507,)
(507, 147)
(168,)
(168, 147)
```

d) Scale all features / predictors (NOT THE TARGET VARIABLE) Feel free to use the sklearn tool "StandardScaler" - more info here: http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html (http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html) (Links to an external site.)

Note: We need to scale here due to SVM. Please refer to previous assignments if you have forgotten appropriate scaling.

In [5]:

```
# Fit standard scaler on X_train data and transform both X_train and X_test data
stscaler = StandardScaler().fit(X_train)
X_train_scaled = stscaler.transform(X_train)
X_test_scaled = stscaler.transform(X_test)
```

Part II. Random Forest Classifier - Base Model

Start by creating a simple Random Forest only using default parameters - this will let us compare SVMs to Random Forest in multiclass problems.

a) Use the RandomForestClassifier in sklearn. Fit your model on the training data.

In [6]:

```
# Create a Random Forest instance, set random_state = SEED for reproducibility
rf = RandomForestClassifier(random_state = SEED)
# Fit on X_train_scaled and y_train
rf.fit(X_train_scaled, y_train)
```

Out[6]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=Non
e,
                       criterion='gini', max_depth=None, max_features
='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=N
one,
                       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=3, v
erbose=0,
                       warm_start=False)
```

b) Use the fitted model to predict on test data. Use the .predict() method to get the predicted classes.

In [7]:

```
# Predict test data
y_pred_rf = rf.predict(X_test_scaled)
```

c) Calculate the confusion matrix and classification report for the test data.

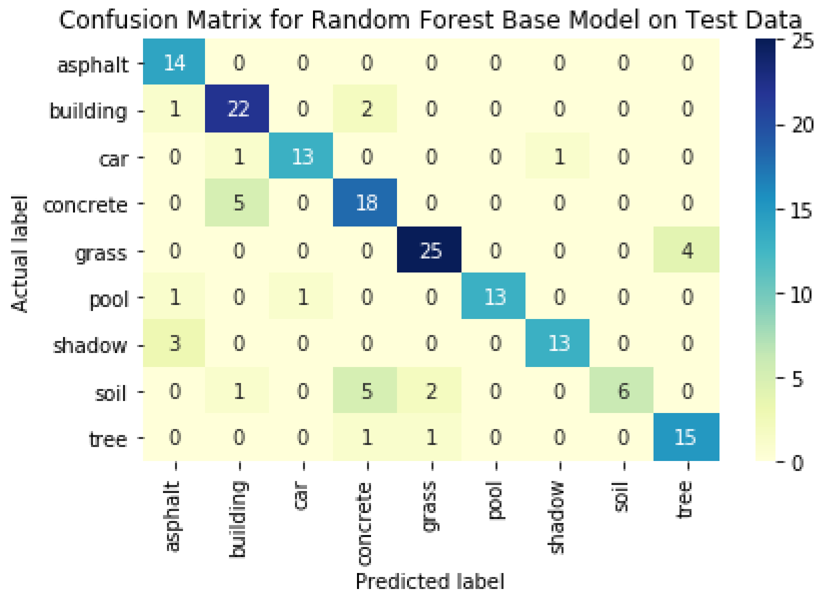
In [8]:

```
# Generate confusion matrix and classification report
cnf_rf_test = metrics.confusion_matrix(y_test, y_pred_rf)
clr_rf_test = metrics.classification_report(y_test, y_pred_rf)

# Put confusion matrix into a dataframe with labels for plotting
class_names = ['asphalt','building','car','concrete','grass','pool','shadow','soil',
df_cnf_rf_test = pd.DataFrame(cnf_rf_test, index=class_names, columns=class_names)

# Create Heatmap for Confusion Matrix
sns.heatmap(df_cnf_rf_test, annot=True, cmap="YlGnBu" ,fmt='g')
plt.tight_layout()
plt.title('Confusion Matrix for Random Forest Base Model on Test Data', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')

# Fix for mpl bug that cuts off top/bottom of seaborn visualization
b, t = plt.ylim() # discover the values for bottom and top
b += 0.5 # Add 0.5 to the bottom
t -= 0.5 # Subtract 0.5 from the top
plt.ylim(b, t) # update the ylim(bottom, top) values
plt.show()
```



	asphalt	building	car	concrete	grass	pool	shadow	soil	tree
asphalt	14	0	0	0	0	0	0	0	0
building	1	22	0	2	0	0	0	0	0
car	0	1	13	0	0	0	1	0	0
concrete	0	5	0	18	0	0	0	0	0
grass	0	0	0	0	25	0	0	0	4
pool	1	0	1	0	0	13	0	0	0
shadow	3	0	0	0	0	0	13	0	0
soil	0	1	0	5	2	0	0	6	0
tree	0	0	0	1	1	0	0	0	15

In [9]:

```
# Print classification report
print(clr_rf_test)
```

```
              precision    recall  f1-score   support

     asphalt       0.74      1.00      0.85        14
    building       0.76      0.88      0.81        25
         car       0.93      0.87      0.90        15
    concrete       0.69      0.78      0.73        23
       grass       0.89      0.86      0.88        29
        pool       1.00      0.87      0.93        15
      shadow       0.93      0.81      0.87        16
        soil       1.00      0.43      0.60        14
        tree       0.79      0.88      0.83        17

    accuracy                           0.83       168
   macro avg       0.86      0.82      0.82       168
weighted avg       0.85      0.83      0.82       168
```

d) Calculate predictions for the training data & build the classification report & confusion matrix. Are there signs of overfitting? Why or why not?

In [10]:

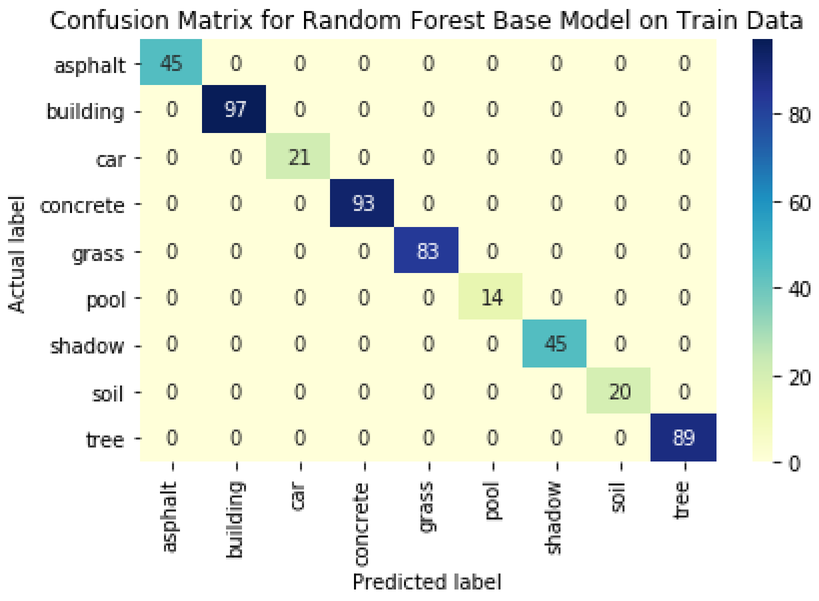
```
# Predict train data
y_pred_rf_train = rf.predict(X_train_scaled)

# Generate confusion matrix
cnf_rf_train = metrics.confusion_matrix(y_train, y_pred_rf_train)

# Put confusion matrix into a dataframe with labels for plotting (class_names speci
df_cnf_rf_train = pd.DataFrame(cnf_rf_train, index=class_names, columns=class_names)

# Create Heatmap for Confusion Matrix
sns.heatmap(df_cnf_rf_train, annot=True, cmap="YlGnBu" ,fmt='g')
plt.tight_layout()
plt.title('Confusion Matrix for Random Forest Base Model on Train Data', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')

# Fix for mpl bug that cuts off top/bottom of seaborn visualization
b, t = plt.ylim() # discover the values for bottom and top
b += 0.5 # Add 0.5 to the bottom
t -= 0.5 # Subtract 0.5 from the top
plt.ylim(b, t) # update the ylim(bottom, top) values
plt.show()
```



	asphalt	building	car	concrete	grass	pool	shadow	soil	tree
asphalt	45	0	0	0	0	0	0	0	0
building	0	97	0	0	0	0	0	0	0
car	0	0	21	0	0	0	0	0	0
concrete	0	0	0	93	0	0	0	0	0
grass	0	0	0	0	83	0	0	0	0
pool	0	0	0	0	0	14	0	0	0
shadow	0	0	0	0	0	0	45	0	0
soil	0	0	0	0	0	0	0	20	0
tree	0	0	0	0	0	0	0	0	89

In [11]:

```
# Generate classification report
clr_rf_train = metrics.classification_report(y_train, y_pred_rf_train)
print(clr_rf_train)
```

```
              precision    recall  f1-score   support

     asphalt       1.00      1.00      1.00        45
    building       1.00      1.00      1.00        97
         car       1.00      1.00      1.00        21
    concrete       1.00      1.00      1.00        93
       grass       1.00      1.00      1.00        83
        pool       1.00      1.00      1.00        14
      shadow       1.00      1.00      1.00        45
        soil       1.00      1.00      1.00        20
        tree       1.00      1.00      1.00        89

    accuracy                           1.00       507
   macro avg       1.00      1.00      1.00       507
weighted avg       1.00      1.00      1.00       507
```

There are signs of overfitting since the accuracy of the train data (1.00) are higher than that of the test data (0.83). The classification reports of the train data suggest perfect classification for all classes, significantly surpassing the precision, recall and f1-score for each class in the test data. These signs suggest that the model has learned the pattern from the train data extremely well, including the noise, and thus the pattern in the train data cannot be applied to the test data and fail to achieve the same level of accuracy when making predictions on the test data.

e) Identify the top 5 features. Feel free to print a list OR to make a plot.

In [12]:

```
# Understand feature importance
importances_rf = pd.DataFrame({'feature':X_train.columns,'importance':np.round(rf.fe
importances_rf = importances_rf.sort_values('importance',ascending=False).set_index(
importances_rf.head(5)
```

Out[12]:

	importance
feature	
NDVI	0.041
NDVI_40	0.031
Mean_R	0.030
Mean_NIR	0.029
Mean_R_40	0.026

Part III. LinearSVM Classifier - Base Model

Create a simple LinearSVC Classifier only using default parameters.

a) Use the LinearSVC in sklearn. Fit your model on the training data.

In [13]:

```
# Create a LinearSVC instance, set random_state = SEED for reproducibility
lsvc = LinearSVC(random_state = SEED)
# Fit on X_train_scaled and y_train
lsvc.fit(X_train_scaled, y_train)
```

Out[13]:

```
LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=3, tol=0.0001,
          verbose=0)
```

b) Use the fitted model to predict on test data. Use the .predict() method to get the predicted classes.

In [14]:

```
# Predict test data
y_pred_lsvc = lsvc.predict(X_test_scaled)
```

c) Calculate the confusion matrix and classification report for test data.

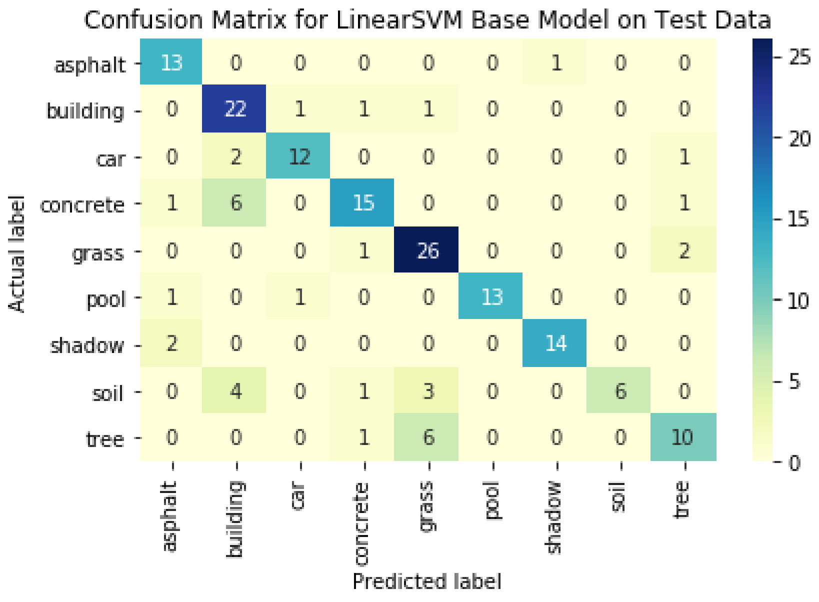
In [15]:

```
# Generate confusion matrix and classification report
cnf_lsvc_test = metrics.confusion_matrix(y_test, y_pred_lsvc)
clr_lsvc_test = metrics.classification_report(y_test, y_pred_lsvc)

# Put confusion matrix into a dataframe with labels for plotting
df_cnf_lsvc_test = pd.DataFrame(cnf_lsvc_test, index=class_names, columns=class_name

# Create Heatmap for Confusion Matrix
sns.heatmap(df_cnf_lsvc_test, annot=True, cmap="YlGnBu" ,fmt='g')
plt.tight_layout()
plt.title('Confusion Matrix for LinearSVM Base Model on Test Data', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')

# Fix for mpl bug that cuts off top/bottom of seaborn visualization
b, t = plt.ylim() # discover the values for bottom and top
b += 0.5 # Add 0.5 to the bottom
t -= 0.5 # Subtract 0.5 from the top
plt.ylim(b, t) # update the ylim(bottom, top) values
plt.show()
```


Confusion Matrix for LinearSVM Base Model on Test Data

Actual label \ Predicted label	asphalt	building	car	concrete	grass	pool	shadow	soil	tree
asphalt	13	0	0	0	0	0	1	0	0
building	0	22	1	1	1	0	0	0	0
car	0	2	12	0	0	0	0	0	1
concrete	1	6	0	15	0	0	0	0	1
grass	0	0	0	1	26	0	0	0	2
pool	1	0	1	0	0	13	0	0	0
shadow	2	0	0	0	0	0	14	0	0
soil	0	4	0	1	3	0	0	6	0
tree	0	0	0	1	6	0	0	0	10

In [16]:

```
# Print classification report
print(clr_lsvc_test)
```

```
              precision    recall  f1-score   support

     asphalt       0.76      0.93      0.84        14
    building       0.65      0.88      0.75        25
         car       0.86      0.80      0.83        15
    concrete       0.79      0.65      0.71        23
       grass       0.72      0.90      0.80        29
        pool       1.00      0.87      0.93        15
      shadow       0.93      0.88      0.90        16
        soil       1.00      0.43      0.60        14
        tree       0.71      0.59      0.65        17

    accuracy                           0.78       168
   macro avg       0.83      0.77      0.78       168
weighted avg       0.80      0.78      0.77       168
```

d) Calculate predictions for the training data & build the classification report & confusion matrix. Are there signs of overfitting? Why or why not?

In [17]:

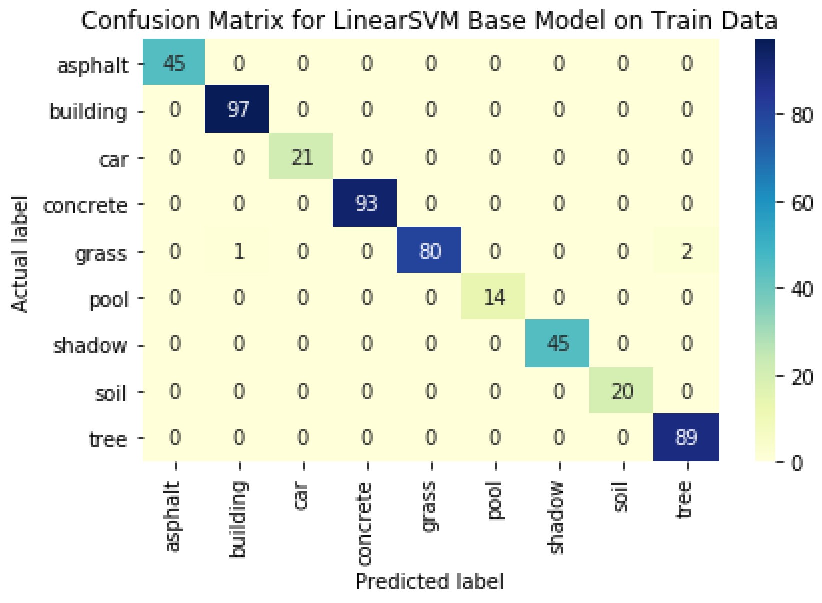
```
# Predict train data
y_pred_lsvc_train = lsvc.predict(X_train_scaled)

# Generate confusion matrix
cnf_lsvc_train = metrics.confusion_matrix(y_train, y_pred_lsvc_train)

# Put confusion matrix into a dataframe with labels for plotting
df_cnf_lsvc_train = pd.DataFrame(cnf_lsvc_train, index=class_names, columns=class_na

# Create Heatmap for Confusion Matrix
sns.heatmap(df_cnf_lsvc_train, annot=True, cmap="YlGnBu" ,fmt='g')
plt.tight_layout()
plt.title('Confusion Matrix for LinearSVM Base Model on Train Data', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')

# Fix for mpl bug that cuts off top/bottom of seaborn visualization
b, t = plt.ylim() # discover the values for bottom and top
b += 0.5 # Add 0.5 to the bottom
t -= 0.5 # Subtract 0.5 from the top
plt.ylim(b, t) # update the ylim(bottom, top) values
plt.show()
```



	asphalt	building	car	concrete	grass	pool	shadow	soil	tree
asphalt	45	0	0	0	0	0	0	0	0
building	0	97	0	0	0	0	0	0	0
car	0	0	21	0	0	0	0	0	0
concrete	0	0	0	93	0	0	0	0	0
grass	0	1	0	0	80	0	0	0	2
pool	0	0	0	0	0	14	0	0	0
shadow	0	0	0	0	0	0	45	0	0
soil	0	0	0	0	0	0	0	20	0
tree	0	0	0	0	0	0	0	0	89

In [18]:

```
# Generate classification report
clr_lsvc_train = metrics.classification_report(y_train, y_pred_lsvc_train)
print(clr_lsvc_train)
```

```
              precision    recall  f1-score   support

     asphalt       1.00      1.00      1.00        45
    building       0.99      1.00      0.99        97
         car       1.00      1.00      1.00        21
    concrete       1.00      1.00      1.00        93
       grass       1.00      0.96      0.98        83
        pool       1.00      1.00      1.00        14
      shadow       1.00      1.00      1.00        45
        soil       1.00      1.00      1.00        20
        tree       0.98      1.00      0.99        89

    accuracy                           0.99       507
   macro avg       1.00      1.00      1.00       507
weighted avg       0.99      0.99      0.99       507
```

There are signs of overfitting since the accuracy of the train data (0.99) is higher than that of the test data (0.78). The classification reports of the train data suggest almost perfect classification for all classes, significantly surpassing the precision, recall and f1-score for each class in the test data. These signs suggest that the model has learned the pattern from the train data extremely well, including the noise, and thus the pattern in the train data cannot be applied to the test data and fail to achieve the same level of accuracy when making predictions on the test data.

Part IV. Supprt Vector Machine Classifier + Linear Kernel + GridSearch

We will now use GridSearchCV to try various hyperparameters in a SVM with linear kernel.

a) Use SVC from sklearn with kernel = "linear". Run the GridSearchCV using the following (SVMs run much faster than RandomForest):

C: 0.01 - 10 in increments of 0.2 (consider using the np.arange() method from numpy to build out a sequence of values)

Note: Feel free to try out more parameters, the above is the bare minimum for this assignment.

Use 5 cross-fold and the default scoring. Please set verbose = 0 to reduce the printing (sorry to our grader for not specifying this last week!).

In [19]:

```
# Create a SVC instance with kernel = linear, set random_state = SEED for reproducil
svc_lin = SVC(kernel = "linear", random_state = SEED)

# Create a dictionary of parameters
param_grid1 = {'C': np.arange(0.01, 10.01, 0.2)}

# Create grid search object with various combinations of parameters
svc_lin_Grid = GridSearchCV(svc_lin, param_grid1, cv = 5, scoring = 'accuracy', ref:

# Fit the grid search model on training data
svc_lin_Grid.fit(X_train_scaled, y_train)
```

Out[19]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=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=3, shrinkin
g=True,
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'C': array([0.01, 0.21, 0.41, 0.61, 0.81, 1.0
1, 1.21, 1.41, 1.61, 1.81, 2.01,
       2.21, 2.41, 2.61, 2.81, 3.01, 3.21, 3.41, 3.61, 3.81, 4.01, 4.2
1,
       4.41, 4.61, 4.81, 5.01, 5.21, 5.41, 5.61, 5.81, 6.01, 6.21, 6.4
1,
       6.61, 6.81, 7.01, 7.21, 7.41, 7.61, 7.81, 8.01, 8.21, 8.41, 8.6
1,
       8.81, 9.01, 9.21, 9.41, 9.61, 9.81])},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=F
alse,
             scoring='accuracy', verbose=0)
```

b) Identify the best performing model:

- .best_params_() : This method outputs to best performing parameters
- .best_estimator_() : This method outputs the best performing model, and can be used for predicting on the X_test

In [20]:

```
# Identify the best model from my grid search
best_svclin = svc_lin_Grid.best_estimator_
best_svclin
```

Out[20]:

```
SVC(C=0.01, break_ties=False, cache_size=200, class_weight=None, coef0
=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='li
near',
    max_iter=-1, probability=False, random_state=3, shrinking=True, to
l=0.001,
    verbose=False)
```

c) Use the best estimator model to predict on test data. Use the .predict() method to get the predicted classes.

In [21]:

```
# Predict test data using best model
y_pred_svclin = best_svclin.predict(X_test_scaled)
```

d) Calculate the confusion matrix and classification report for test data.

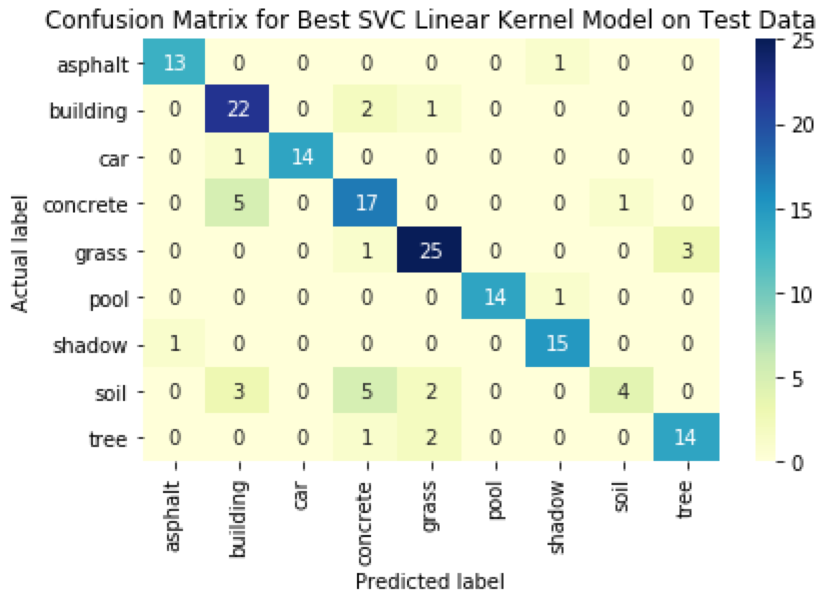
In [22]:

```
# Generate confusion matrix and classification report
cnf_svclin_test = metrics.confusion_matrix(y_test, y_pred_svclin)
clr_svclin_test = metrics.classification_report(y_test, y_pred_svclin)

# Put confusion matrix into a dataframe with labels for plotting
df_cnf_svclin_test = pd.DataFrame(cnf_svclin_test, index=class_names, columns=class_

# Create Heatmap for Confusion Matrix
sns.heatmap(df_cnf_svclin_test, annot=True, cmap="YlGnBu" ,fmt='g')
plt.tight_layout()
plt.title('Confusion Matrix for Best SVC Linear Kernel Model on Test Data', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')

# Fix for mpl bug that cuts off top/bottom of seaborn visualization
b, t = plt.ylim() # discover the values for bottom and top
b += 0.5 # Add 0.5 to the bottom
t -= 0.5 # Subtract 0.5 from the top
plt.ylim(b, t) # update the ylim(bottom, top) values
plt.show()
```



	asphalt	building	car	concrete	grass	pool	shadow	soil	tree
asphalt	13	0	0	0	0	0	1	0	0
building	0	22	0	2	1	0	0	0	0
car	0	1	14	0	0	0	0	0	0
concrete	0	5	0	17	0	0	0	1	0
grass	0	0	0	1	25	0	0	0	3
pool	0	0	0	0	0	14	1	0	0
shadow	1	0	0	0	0	0	15	0	0
soil	0	3	0	5	2	0	0	4	0
tree	0	0	0	1	2	0	0	0	14

In [23]:

```
# Print classification report
print(clr_svclin_test)
```

```
              precision    recall  f1-score   support

     asphalt       0.93      0.93      0.93        14
    building       0.71      0.88      0.79        25
         car       1.00      0.93      0.97        15
    concrete       0.65      0.74      0.69        23
       grass       0.83      0.86      0.85        29
        pool       1.00      0.93      0.97        15
      shadow       0.88      0.94      0.91        16
        soil       0.80      0.29      0.42        14
        tree       0.82      0.82      0.82        17

    accuracy                           0.82       168
   macro avg       0.85      0.81      0.82       168
weighted avg       0.83      0.82      0.81       168
```

e) Calculate predictions for the training data & build the classification report & confusion matrix. Are there signs of overfitting? Why or why not?

In [24]:

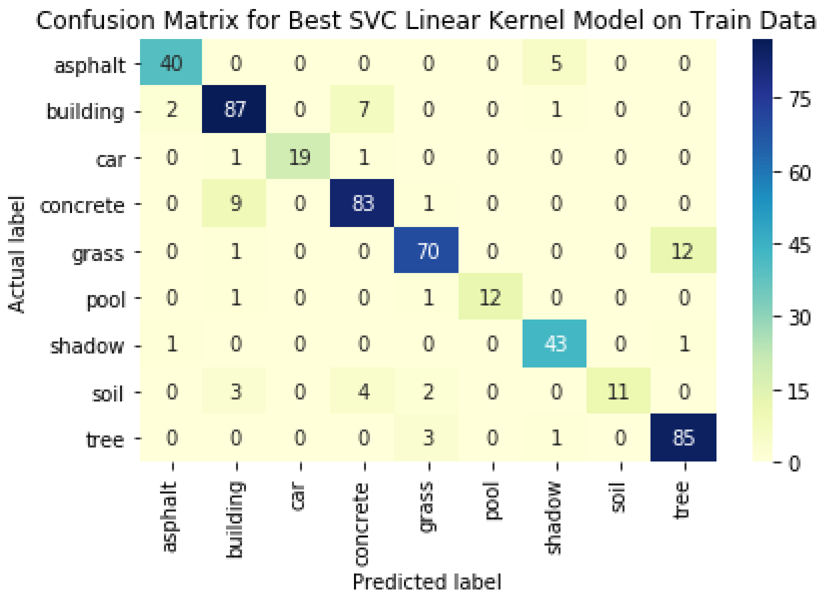
```
# Predict train data
y_pred_svclin_train = best_svclin.predict(X_train_scaled)

# Generate confusion matrix
cnf_svclin_train = metrics.confusion_matrix(y_train, y_pred_svclin_train)

# Put confusion matrix into a dataframe with labels for plotting
df_cnf_svclin_train = pd.DataFrame(cnf_svclin_train, index=class_names, columns=clas

# Create Heatmap for Confusion Matrix
sns.heatmap(df_cnf_svclin_train, annot=True, cmap="YlGnBu" ,fmt='g')
plt.tight_layout()
plt.title('Confusion Matrix for Best SVC Linear Kernel Model on Train Data', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')

# Fix for mpl bug that cuts off top/bottom of seaborn visualization
b, t = plt.ylim() # discover the values for bottom and top
b += 0.5 # Add 0.5 to the bottom
t -= 0.5 # Subtract 0.5 from the top
plt.ylim(b, t) # update the ylim(bottom, top) values
plt.show()
```



	asphalt	building	car	concrete	grass	pool	shadow	soil	tree
asphalt	40	0	0	0	0	0	5	0	0
building	2	87	0	7	0	0	1	0	0
car	0	1	19	1	0	0	0	0	0
concrete	0	9	0	83	1	0	0	0	0
grass	0	1	0	0	70	0	0	0	12
pool	0	1	0	0	1	12	0	0	0
shadow	1	0	0	0	0	0	43	0	1
soil	0	3	0	4	2	0	0	11	0
tree	0	0	0	0	3	0	1	0	85

In [25]:

```
# Generate classification report
clr_svclin_train = metrics.classification_report(y_train, y_pred_svclin_train)
print(clr_svclin_train)
```

```
              precision    recall  f1-score   support

     asphalt       0.93      0.89      0.91        45
    building       0.85      0.90      0.87        97
         car       1.00      0.90      0.95        21
    concrete       0.87      0.89      0.88        93
       grass       0.91      0.84      0.88        83
        pool       1.00      0.86      0.92        14
      shadow       0.86      0.96      0.91        45
        soil       1.00      0.55      0.71        20
        tree       0.87      0.96      0.91        89

    accuracy                           0.89       507
   macro avg       0.92      0.86      0.88       507
weighted avg       0.89      0.89      0.89       507
```

There are signs of slight overfitting since the accuracy of the train data (0.89) is a bit higher than that of the test data (0.82). By taking a closer look at the classification reports, we can see that the precision, recall and f1-score between the train and test data come quite close, except for 3 classes - building, concrete, soil - where the metrics for test data are lower than those for the train data. While the model performs better on the train data, which is a sign of overfitting, it suffers from the mildest overfitting issue among all the models and that's why the difference between predictions on train and test data are quite small.

Part V. Support Vector Machine Classifier + Polynomial Kernel + Grid Search

We will now use GridSearchCV to try various hyperparameters in a SVM with a polynomial kernel.

a) Use SVC from sklearn with kernel = "poly". Run the GridSearchCV using the following:

- C: 0.01 - 10 in increments of 0.2
- degree: 2, 3, 4, 5, 6

Note: Feel free to try out more parameters, the above is the bare minimum for this assignment.

Use 5 cross-fold and the default scoring.

In [26]:

```
# Create a SVC instance with kernel = poly, set random_state = SEED for reproducibi
svc_poly = SVC(kernel = "poly", random_state = SEED)

# Create a dictionary of parameters
param_grid2 = {'C': np.arange(0.01, 10.01, 0.2),
               'degree': [2, 3, 4, 5, 6]}

# Create grid search object with various combinations of parameters
svc_poly_Grid = GridSearchCV(svc_poly, param_grid2, cv = 5, scoring = 'accuracy', re

# Fit the grid search model on training data
svc_poly_Grid.fit(X_train_scaled, y_train)
```

Out[26]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=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='poly', max_iter=-1,
                           probability=False, random_state=3, shrinkin
g=True,
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'C': array([0.01, 0.21, 0.41, 0.61, 0.81, 1.0
1, 1.21, 1.41, 1.61, 1.81, 2.01,
       2.21, 2.41, 2.61, 2.81, 3.01, 3.21, 3.41, 3.61, 3.81, 4.01, 4.2
1,
       4.41, 4.61, 4.81, 5.01, 5.21, 5.41, 5.61, 5.81, 6.01, 6.21, 6.4
1,
       6.61, 6.81, 7.01, 7.21, 7.41, 7.61, 7.81, 8.01, 8.21, 8.41, 8.6
1,
       8.81, 9.01, 9.21, 9.41, 9.61, 9.81]),
                         'degree': [2, 3, 4, 5, 6]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=F
alse,
             scoring='accuracy', verbose=0)
```

b) Identify the best performing model:

.best_params_() : This method outputs to best performing parameters .best_estimator_() : This method outputs the best performing model, and can be used for predicting on the X_test

In [27]:

```
# Identify the best model from my grid search
best_svcpoly = svc_poly_Grid.best_estimator_
best_svcpoly
```

Out[27]:

```
SVC(C=3.81, break_ties=False, cache_size=200, class_weight=None, coef0
=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='po
ly',
    max_iter=-1, probability=False, random_state=3, shrinking=True, to
l=0.001,
    verbose=False)
```

c) Use the best estimator model to predict on test data. Use the .predict() method to get the predicted classes.

In [28]:

```
# Predict test data using best model
y_pred_svcpoly = best_svcpoly.predict(X_test_scaled)
```

d) Calculate the confusion matrix and classification report for test data.

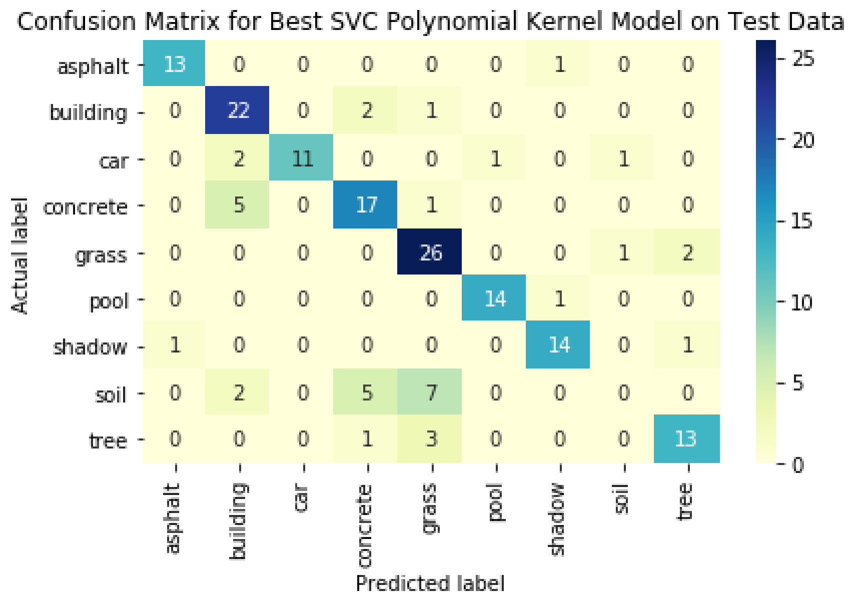
In [29]:

```
# Generate confusion matrix and classification report
cnf_svcpoly_test = metrics.confusion_matrix(y_test, y_pred_svcpoly)
clr_svcpoly_test = metrics.classification_report(y_test, y_pred_svcpoly)

# Put confusion matrix into a dataframe with labels for plotting
df_cnf_svcpoly_test = pd.DataFrame(cnf_svcpoly_test, index=class_names, columns=clas

# Create Heatmap for Confusion Matrix
sns.heatmap(df_cnf_svcpoly_test, annot=True, cmap="YlGnBu" ,fmt='g')
plt.tight_layout()
plt.title('Confusion Matrix for Best SVC Polynomial Kernel Model on Test Data', y=1.
plt.ylabel('Actual label')
plt.xlabel('Predicted label')

# Fix for mpl bug that cuts off top/bottom of seaborn visualization
b, t = plt.ylim() # discover the values for bottom and top
b += 0.5 # Add 0.5 to the bottom
t -= 0.5 # Subtract 0.5 from the top
plt.ylim(b, t) # update the ylim(bottom, top) values
plt.show()
```



	asphalt	building	car	concrete	grass	pool	shadow	soil	tree
asphalt	13	0	0	0	0	0	1	0	0
building	0	22	0	2	1	0	0	0	0
car	0	2	11	0	0	1	0	1	0
concrete	0	5	0	17	1	0	0	0	0
grass	0	0	0	0	26	0	0	1	2
pool	0	0	0	0	0	14	1	0	0
shadow	1	0	0	0	0	0	14	0	1
soil	0	2	0	5	7	0	0	0	0
tree	0	0	0	1	3	0	0	0	13

In [30]:

```
# Print classification report
print(clr_svcpoly_test)
```

```
              precision    recall  f1-score   support

     asphalt       0.93      0.93      0.93        14
    building       0.71      0.88      0.79        25
         car       1.00      0.73      0.85        15
    concrete       0.68      0.74      0.71        23
       grass       0.68      0.90      0.78        29
        pool       0.93      0.93      0.93        15
      shadow       0.88      0.88      0.88        16
        soil       0.00      0.00      0.00        14
        tree       0.81      0.76      0.79        17

    accuracy                           0.77       168
   macro avg       0.74      0.75      0.74       168
weighted avg       0.73      0.77      0.75       168
```

e) Calculate predictions for the training data & build the classification report & confusion matrix. Are there signs of overfitting? Why or why not?

```
In [31]:
```

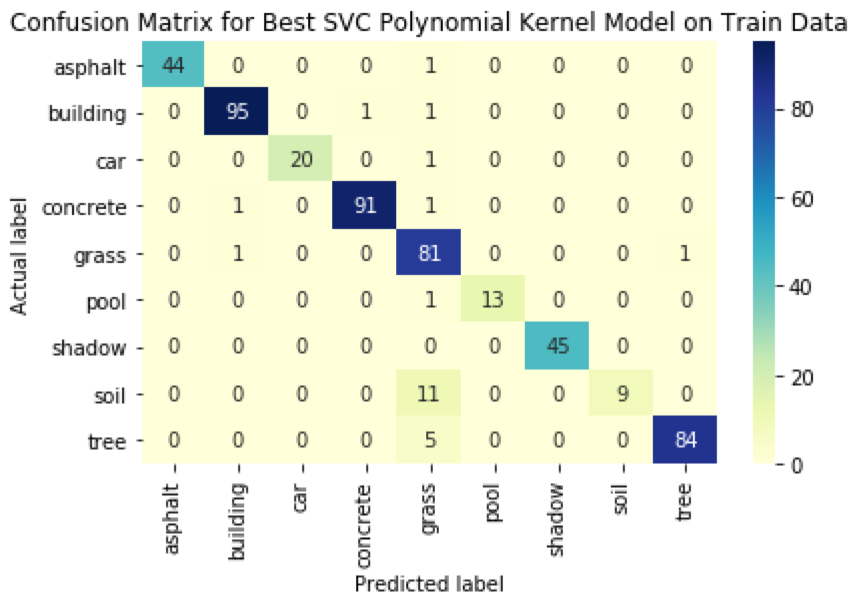
```
# Predict train data
y_pred_svcpoly_train = best_svcpoly.predict(X_train_scaled)

# Generate confusion matrix
cnf_svcpoly_train = metrics.confusion_matrix(y_train, y_pred_svcpoly_train)

# Put confusion matrix into a dataframe with labels for plotting
df_cnf_svcpoly_train = pd.DataFrame(cnf_svcpoly_train, index=class_names, columns=cl

# Create Heatmap for Confusion Matrix
sns.heatmap(df_cnf_svcpoly_train, annot=True, cmap="YlGnBu" ,fmt='g')
plt.tight_layout()
plt.title('Confusion Matrix for Best SVC Polynomial Kernel Model on Train Data', y=
plt.ylabel('Actual label')
plt.xlabel('Predicted label')

# Fix for mpl bug that cuts off top/bottom of seaborn visualization
b, t = plt.ylim() # discover the values for bottom and top
b += 0.5 # Add 0.5 to the bottom
t -= 0.5 # Subtract 0.5 from the top
plt.ylim(b, t) # update the ylim(bottom, top) values
plt.show()
```



Actual \ Predicted	asphalt	building	car	concrete	grass	pool	shadow	soil	tree
asphalt	44	0	0	0	1	0	0	0	0
building	0	95	0	1	1	0	0	0	0
car	0	0	20	0	1	0	0	0	0
concrete	0	1	0	91	1	0	0	0	0
grass	0	1	0	0	81	0	0	0	1
pool	0	0	0	0	1	13	0	0	0
shadow	0	0	0	0	0	0	45	0	0
soil	0	0	0	0	11	0	0	9	0
tree	0	0	0	0	5	0	0	0	84

In [32]:

```python
# Generate classification report
clr_svcpoly_train = metrics.classification_report(y_train, y_pred_svcpoly_train)
print(clr_svcpoly_train)
```

```
              precision    recall  f1-score   support

     asphalt       1.00      0.98      0.99        45
    building       0.98      0.98      0.98        97
         car       1.00      0.95      0.98        21
    concrete       0.99      0.98      0.98        93
       grass       0.79      0.98      0.88        83
        pool       1.00      0.93      0.96        14
      shadow       1.00      1.00      1.00        45
        soil       1.00      0.45      0.62        20
        tree       0.99      0.94      0.97        89

    accuracy                           0.95       507
   macro avg       0.97      0.91      0.93       507
weighted avg       0.96      0.95      0.95       507
```

There are signs of overfitting since the accuracy of the train data (0.95) is higher than that of the test data (0.77). The classification report of the train data reflect higher precision, recall and f1-score overall. These signs suggest that the model has learned the pattern from the train data very well, including the noise, and thus the pattern in the train data cannot be applied to the test data and fail to achieve the same level of accuracy when making predictions on the test data.

Part VI. Support Vector Machine Classifier + RBF Kernel + Grid Search

We will now use GridSearchCV to try various hyperparameters in a SVM with a RBF kernel.

a) Use SVC from sklearn with kernel = "rbf". Run the GridSearchCV using the following:

- C: 0.01 - 10 in increments of 0.2
- gamma: 0.01, 0.1, 1, 10, 100

Note: Feel free to try out more parameters, the above is the bare minimum for this assignment.

Use 5 cross-fold and the default scoring.

In [33]:

```
# Create a SVC instance with kernel = rbf, set random_state = SEED for reproducibil
svc_rbf = SVC(kernel = "rbf", random_state = SEED)

# Create a dictionary of parameters
param_grid3 = {'C': np.arange(0.01, 10.01, 0.2),
               'gamma': [0.01, 0.1, 1, 10, 100]}

# Create grid search object with various combinations of parameters
svc_rbf_Grid = GridSearchCV(svc_rbf, param_grid3, cv = 5, scoring = 'accuracy', ref

# Fit the grid search model on training data
svc_rbf_Grid.fit(X_train_scaled, y_train)
```

Out[33]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=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='rbf', max_iter=-1,
                           probability=False, random_state=3, shrinkin
g=True,
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'C': array([0.01, 0.21, 0.41, 0.61, 0.81, 1.0
1, 1.21, 1.41, 1.61, 1.81, 2.01,
       2.21, 2.41, 2.61, 2.81, 3.01, 3.21, 3.41, 3.61, 3.81, 4.01, 4.2
1,
       4.41, 4.61, 4.81, 5.01, 5.21, 5.41, 5.61, 5.81, 6.01, 6.21, 6.4
1,
       6.61, 6.81, 7.01, 7.21, 7.41, 7.61, 7.81, 8.01, 8.21, 8.41, 8.6
1,
       8.81, 9.01, 9.21, 9.41, 9.61, 9.81]),
                         'gamma': [0.01, 0.1, 1, 10, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=F
alse,
             scoring='accuracy', verbose=0)
```

b) Identify the best performing model:

.best_params_() : This method outputs to best performing parameters .best_estimator_() : This method outputs the best performing model, and can be used for predicting on the X_test

In [34]:

```
# Identify the best model from my grid search
best_svcrbf = svc_rbf_Grid.best_estimator_
best_svcrbf
```

Out[34]:

```
SVC(C=2.81, break_ties=False, cache_size=200, class_weight=None, coef0
=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.01, kernel='rbf',
    max_iter=-1, probability=False, random_state=3, shrinking=True, to
l=0.001,
    verbose=False)
```

c) Use the best estimator model to predict on test data. Use the .predict() method to get the predicted classes.

In [35]:

```
# Predict test data using best model
y_pred_svcrbf = best_svcrbf.predict(X_test_scaled)
```

d) Calculate the confusion matrix and classification report for test data.

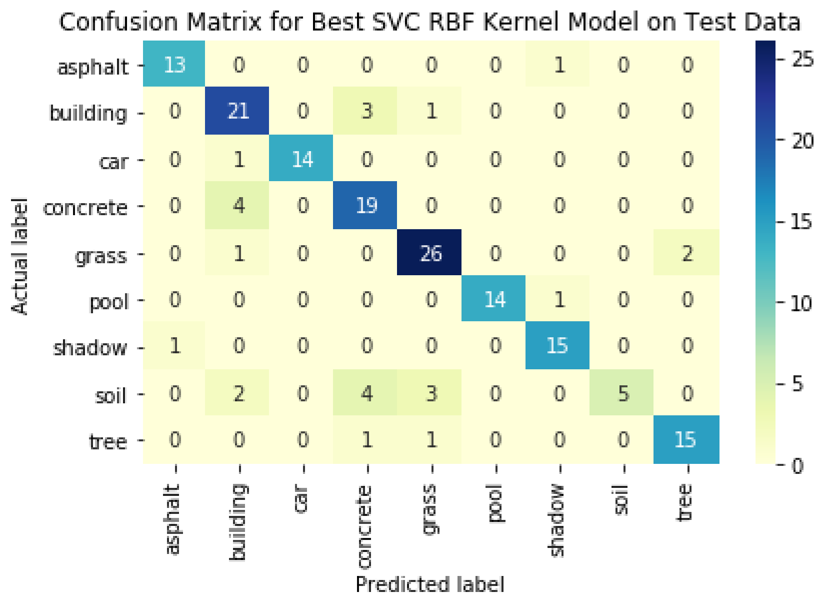
In [36]:

```
# Generate confusion matrix and classification report
cnf_svcrbf_test = metrics.confusion_matrix(y_test, y_pred_svcrbf)
clr_svcrbf_test = metrics.classification_report(y_test, y_pred_svcrbf)

# Put confusion matrix into a dataframe with labels for plotting
df_cnf_svcrbf_test = pd.DataFrame(cnf_svcrbf_test, index=class_names, columns=class_

# Create Heatmap for Confusion Matrix
sns.heatmap(df_cnf_svcrbf_test, annot=True, cmap="YlGnBu" ,fmt='g')
plt.tight_layout()
plt.title('Confusion Matrix for Best SVC RBF Kernel Model on Test Data', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')

# Fix for mpl bug that cuts off top/bottom of seaborn visualization
b, t = plt.ylim() # discover the values for bottom and top
b += 0.5 # Add 0.5 to the bottom
t -= 0.5 # Subtract 0.5 from the top
plt.ylim(b, t) # update the ylim(bottom, top) values
plt.show()
```



	asphalt	building	car	concrete	grass	pool	shadow	soil	tree
asphalt	13	0	0	0	0	0	1	0	0
building	0	21	0	3	1	0	0	0	0
car	0	1	14	0	0	0	0	0	0
concrete	0	4	0	19	0	0	0	0	0
grass	0	1	0	0	26	0	0	0	2
pool	0	0	0	0	0	14	1	0	0
shadow	1	0	0	0	0	0	15	0	0
soil	0	2	0	4	3	0	0	5	0
tree	0	0	0	1	1	0	0	0	15

In [37]:

```
# Print classification report
print(clr_svcrbf_test)
```

```
              precision    recall  f1-score   support

     asphalt       0.93      0.93      0.93        14
    building       0.72      0.84      0.78        25
         car       1.00      0.93      0.97        15
    concrete       0.70      0.83      0.76        23
       grass       0.84      0.90      0.87        29
        pool       1.00      0.93      0.97        15
      shadow       0.88      0.94      0.91        16
        soil       1.00      0.36      0.53        14
        tree       0.88      0.88      0.88        17

    accuracy                           0.85       168
   macro avg       0.88      0.84      0.84       168
weighted avg       0.86      0.85      0.84       168
```

e) Calculate predictions for the training data & build the classification report & confusion matrix. Are there signs of overfitting? Why or why not?

In [38]:

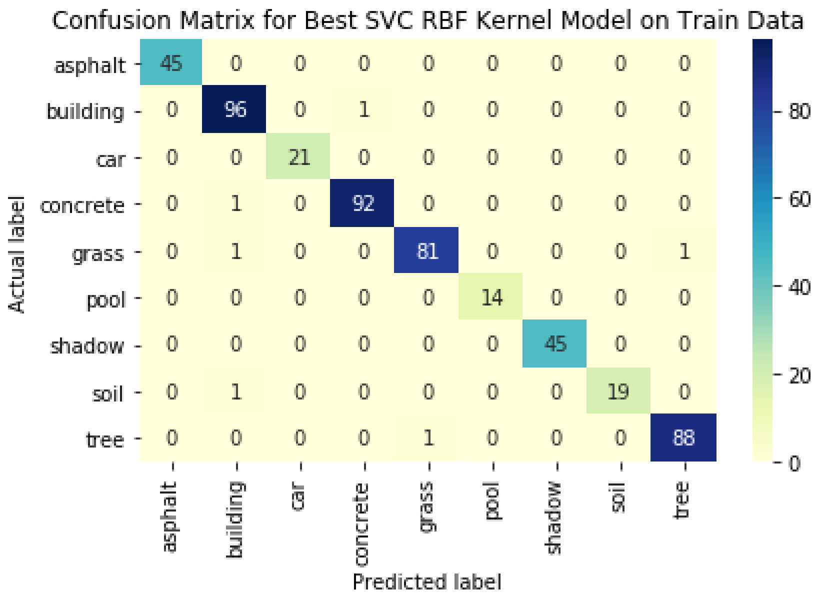
```
# Predict train data
y_pred_svcrbf_train = best_svcrbf.predict(X_train_scaled)

# Generate confusion matrix
cnf_svcrbf_train = metrics.confusion_matrix(y_train, y_pred_svcrbf_train)

# Put confusion matrix into a dataframe with labels for plotting
df_cnf_svcrbf_train = pd.DataFrame(cnf_svcrbf_train, index=class_names, columns=clas

# Create Heatmap for Confusion Matrix
sns.heatmap(df_cnf_svcrbf_train, annot=True, cmap="YlGnBu" ,fmt='g')
plt.tight_layout()
plt.title('Confusion Matrix for Best SVC RBF Kernel Model on Train Data', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')

# Fix for mpl bug that cuts off top/bottom of seaborn visualization
b, t = plt.ylim() # discover the values for bottom and top
b += 0.5 # Add 0.5 to the bottom
t -= 0.5 # Subtract 0.5 from the top
plt.ylim(b, t) # update the ylim(bottom, top) values
plt.show()
```



	asphalt	building	car	concrete	grass	pool	shadow	soil	tree
asphalt	45	0	0	0	0	0	0	0	0
building	0	96	0	1	0	0	0	0	0
car	0	0	21	0	0	0	0	0	0
concrete	0	1	0	92	0	0	0	0	0
grass	0	1	0	0	81	0	0	0	1
pool	0	0	0	0	0	14	0	0	0
shadow	0	0	0	0	0	0	45	0	0
soil	0	1	0	0	0	0	0	19	0
tree	0	0	0	0	1	0	0	0	88

In [39]:

```
# Generate classification report
clr_svcrbf_train = metrics.classification_report(y_train, y_pred_svcrbf_train)
print(clr_svcrbf_train)
```

```
              precision    recall  f1-score   support

     asphalt       1.00      1.00      1.00        45
    building       0.97      0.99      0.98        97
         car       1.00      1.00      1.00        21
    concrete       0.99      0.99      0.99        93
       grass       0.99      0.98      0.98        83
        pool       1.00      1.00      1.00        14
      shadow       1.00      1.00      1.00        45
        soil       1.00      0.95      0.97        20
        tree       0.99      0.99      0.99        89

    accuracy                           0.99       507
   macro avg       0.99      0.99      0.99       507
weighted avg       0.99      0.99      0.99       507
```

There are signs of overfitting since the accuracy of the train data (0.99) is higher than that of the test data (0.85). The classification report of the train data reflect higher precision, recall and f1-score overall. These signs suggest that the model has learned the pattern from the train data very well, including the noise, and thus the pattern in the train data cannot be applied to the test data and fail to achieve the same level of accuracy when making predictions on the test data.

Part VII. Conceptual Questions

a) From the models run in steps 2-6, which performs the best based on the Classification Report? Support your reasoning with evidence around your test data.

In [40]:

```
print("Classification Report of Random Forest Base Model on Test Data")
print(clr_rf_test)
print("*"*60)
print("Classification Report of LinearSVM Base Model on Test Data")
print(clr_lsvc_test)
print("*"*60)
print("Classification Report of Best SVC Linear Kernel Model on Test Data")
print(clr_svclin_test)
print("*"*60)
print("Classification Report of Best SVC Polynomial Kernel Model on Test Data")
print(clr_svcpoly_test)
print("*"*60)
print("Classification Report of Best SVC RBF Kernel Model on Test Data")
print(clr_svcrbf_test)
```

```
Classification Report of Random Forest Base Model on Test Data
              precision    recall  f1-score   support

     asphalt       0.74      1.00      0.85        14
    building       0.76      0.88      0.81        25
         car       0.93      0.87      0.90        15
    concrete       0.69      0.78      0.73        23
       grass       0.89      0.86      0.88        29
        pool       1.00      0.87      0.93        15
      shadow       0.93      0.81      0.87        16
        soil       1.00      0.43      0.60        14
        tree       0.79      0.88      0.83        17

    accuracy                           0.83       168
   macro avg       0.86      0.82      0.82       168
weighted avg       0.85      0.83      0.82       168

************************************************************
Classification Report of LinearSVM Base Model on Test Data
              precision    recall  f1-score   support

     asphalt       0.76      0.93      0.84        14
    building       0.65      0.88      0.75        25
         car       0.86      0.80      0.83        15
    concrete       0.79      0.65      0.71        23
       grass       0.72      0.90      0.80        29
        pool       1.00      0.87      0.93        15
      shadow       0.93      0.88      0.90        16
        soil       1.00      0.43      0.60        14
        tree       0.71      0.59      0.65        17

    accuracy                           0.78       168
   macro avg       0.83      0.77      0.78       168
weighted avg       0.80      0.78      0.77       168

************************************************************
Classification Report of Best SVC Linear Kernel Model on Test Data
              precision    recall  f1-score   support

     asphalt       0.93      0.93      0.93        14
    building       0.71      0.88      0.79        25
         car       1.00      0.93      0.97        15
    concrete       0.65      0.74      0.69        23
       grass       0.83      0.86      0.85        29
```

```
        pool       1.00      0.93      0.97        15
      shadow       0.88      0.94      0.91        16
        soil       0.80      0.29      0.42        14
        tree       0.82      0.82      0.82        17

    accuracy                           0.82       168
   macro avg       0.85      0.81      0.82       168
weighted avg       0.83      0.82      0.81       168

************************************************************
Classification Report of Best SVC Polynomial Kernel Model on Test Data
              precision    recall  f1-score   support

     asphalt       0.93      0.93      0.93        14
    building       0.71      0.88      0.79        25
         car       1.00      0.73      0.85        15
    concrete       0.68      0.74      0.71        23
       grass       0.68      0.90      0.78        29
        pool       0.93      0.93      0.93        15
      shadow       0.88      0.88      0.88        16
        soil       0.00      0.00      0.00        14
        tree       0.81      0.76      0.79        17

    accuracy                           0.77       168
   macro avg       0.74      0.75      0.74       168
weighted avg       0.73      0.77      0.75       168

************************************************************
Classification Report of Best SVC RBF Kernel Model on Test Data
              precision    recall  f1-score   support

     asphalt       0.93      0.93      0.93        14
    building       0.72      0.84      0.78        25
         car       1.00      0.93      0.97        15
    concrete       0.70      0.83      0.76        23
       grass       0.84      0.90      0.87        29
        pool       1.00      0.93      0.97        15
      shadow       0.88      0.94      0.91        16
        soil       1.00      0.36      0.53        14
        tree       0.88      0.88      0.88        17

    accuracy                           0.85       168
   macro avg       0.88      0.84      0.84       168
weighted avg       0.86      0.85      0.84       168
```

The SVC with RBF kernel is the best performing model since it has the highest accuracy and the highest overall precision, recall and f1-score among all the models

b) Compare models run for steps 4-6 where different kernels were used. What is the benefit of using a polynomial or rbf kernel over a linear kernel? What could be a downside of using a polynomial or rbf kernel?

A dataset may not always be linearly separable, i.e. you cannot separate the oberservations into different classes simply by drawing a line as the boundary. Therefore, in addition to the linear kernel, SVM employs other kernel functions to map the original dataset into a higher dimensional space in order to construct a hyperplane decision boundary. Some common kernel functions are shown below:

Equation 5-10. Common kernels

$$\begin{aligned}
\text{Linear:} \quad & K(\mathbf{a}, \mathbf{b}) = \mathbf{a}^T \mathbf{b} \\
\text{Polynomial:} \quad & K(\mathbf{a}, \mathbf{b}) = \left(\gamma \mathbf{a}^T \mathbf{b} + r\right)^d \\
\text{Gaussian RBF:} \quad & K(\mathbf{a}, \mathbf{b}) = \exp\left(-\gamma \| \mathbf{a} - \mathbf{b} \|^2\right) \\
\text{Sigmoid:} \quad & K(\mathbf{a}, \mathbf{b}) = \tanh\left(\gamma \mathbf{a}^T \mathbf{b} + r\right)
\end{aligned}$$

A simple visualization may be helpful. One the left plot, you can see a dataset with one feature X1, which is not linearly separable. However, if you apply a 2nd-degree polynomial kernel function, you can obtain a new feature X2 (where X2 = X1^2) and you are able to map the data to a 2-dimensional space composed of X1 and X2, as shown on the right plot. The 2-D dataset is easily linearly separable.

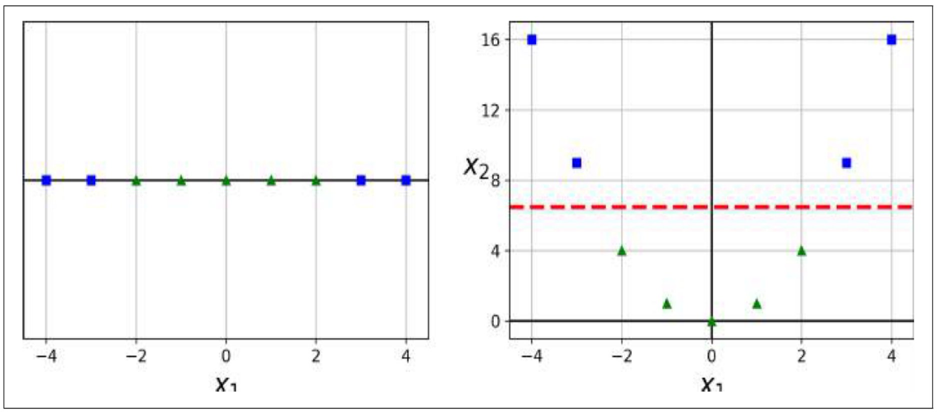


Figure 5-5. Adding features to make a dataset linearly separable

However, there's also a downside of polynomial kernel. It may suffer from numerical instability: when gamma(a.T*b + r) < 1, K(a, b) = gamma(a.T*b + r)^d approaches zero with increasing d. One the other hand, when gamma(a.T*b + r) > 1, K(a, b) approaches infinity as d increases.

c) Explain the 'C' parameter used in steps 4-6. What does a small C mean versus a large C in sklearn? Why is it important to use the 'C' parameter when fitting a model?

C is a regularization paramater that controls the trade-off between minimizing the slack variable to reduce margin violations (which will help us achieving a low error on the training data) and minimizing the norm of the weights (to increase the margin generated by the support vector machine algorithm). The objective function of a linear SVM classifier is shown below:

Equation 5-4. Soft margin linear SVM classifier objective

$$\underset{\mathbf{w}, b, \zeta}{\text{minimize}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{m} \zeta^{(i)}$$

$$\text{subject to} \quad t^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) \geq 1 - \zeta^{(i)} \quad \text{and} \quad \zeta^{(i)} \geq 0 \quad \text{for } i = 1, 2, \cdots, m$$

where the slack variable measures how much the i-th instance is allowed to violate the margin and w represents the weight vector which we want to minimize in order to get a large margin.

If C is too large, the optimization algorithm will try to reduce |w| as much as possible, leading to a hyperplane which tries to classify each training example correctly and cause overfitting. On the other hand, if C is too small then the objective function is given the freedom to increase |w| significantly, which will lead to large training error and result in underfitting. In conclusion, tuning C correctly is a vital step in the use of SVMs as it controls the overfitting and underfitting of the model.

d) Scaling our input data does not matter much for Random Forest, but it is a critical step for Support Vector Machines. Explain why this is such a critical step. Also, provide an example of a feature from this data set that could cause issues with our SVMs if not scaled.

Support Vector Machine is a maximal margin classifier and the algorithm works by maximizing the distance between the separating plane and the support vectors (which are the instances circled in pink on the dotted parallel boundary in the picture below). SVM is sensitive to feature scales as features with a greater scale will have pose a greater influence on the calculation of the margin.

The left plot has X0 and X1 unscaled and the linear decision boundary appears to be horizontal, while the right plot shows X0 and X1 after scaling and we have a slanted decision boundary. We can see that the observations lie at different relative positions in the 2 plots. And if we have more observations, we might find the left decision boundary unable to classify datapoints. This is because X1, the feature with large values, will dominate other features when calculating the distance. If you rescale all features then each feature will have the same influence on the distance metric.

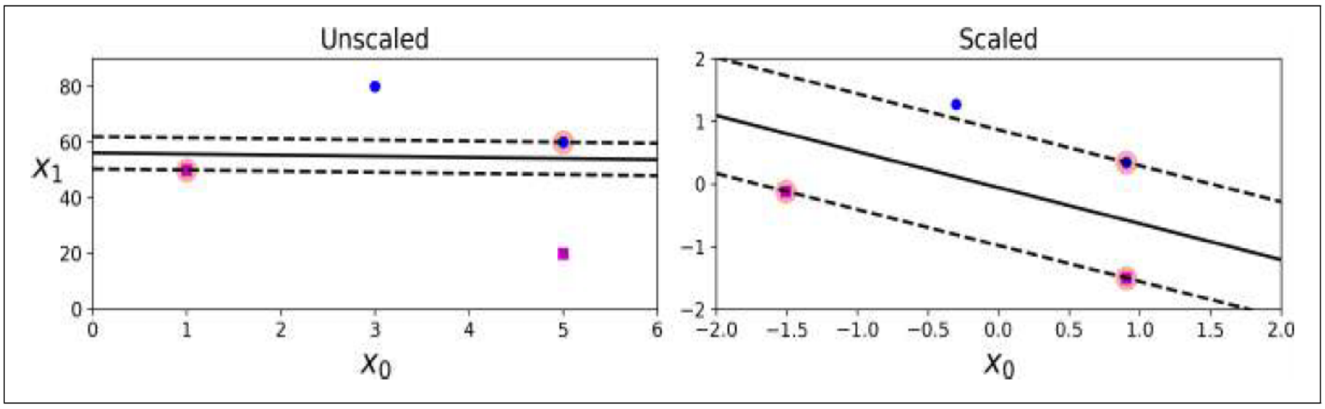


Figure 5-2. Sensitivity to feature scales

The cell below contains the statistical summary of our train data. We can see the Area feature has mean of 562.50 and standard deviation of 698.66, which is much larger in scale compared to variables like Compact, which has a mean of 2.19 and standard deviation of 0.87. If StandardScaler was not performed, the area feature will have much greater influence on the margin calculation than Compact and it could cause issues with our SVM classifier.

In [41]:

```
X_train.describe()
```

Out[41]:

	BrdIndx	Area	Round	Bright	Compact	ShpIndx	Mean_G	
count	507.000000	507.000000	507.000000	507.000000	507.000000	507.000000	507.000000	50
mean	2.025720	562.504931	1.237574	165.612939	2.187081	2.277318	166.290355	16
std	0.619254	698.655240	0.561988	63.230806	0.874054	0.718441	59.217648	7
min	1.000000	22.000000	0.000000	26.850000	1.000000	1.040000	22.910000	2
25%	1.580000	159.000000	0.840000	127.485000	1.650000	1.715000	146.460000	9
50%	1.950000	323.000000	1.210000	170.650000	2.000000	2.180000	189.630000	15
75%	2.380000	681.500000	1.565000	224.825000	2.490000	2.675000	206.780000	23
max	4.530000	5767.000000	3.520000	245.870000	8.070000	5.410000	239.370000	25

8 rows × 147 columns

e) Describe conceptually what the purpose of a kernel is for Support Vector Machines.

The kernel trick enables the mapping of our data into an enlarged feature space for better construction of a hyperplane decision boundary without actually going through the calculation of each vector. In Machine Learning, a kernel is a function capable of computing the dot product ϕ(a)T ϕ(b) based only on the original vectors a and b, without having to compute the transformation ϕ. As such, a kernel is able to define a notion of similarity, with little computational cost even in very high-dimensional spaces, making SVM a powerful algorithm when separating data that are non-linear in nature.