Assignment 5 - Support Vector Machines (Multiclassification)

Matthew Dunne

For assignment #5 we will be working with UCI's Urban Land Cover Data Set. This is the classification of urban land cover with 9 different classes that are fairly balanced across train & test. Multi-scale spectral, size, shape, and texture information are used for classification.

Data: https://archive.ics.uci.edu/ml/datasets/Urban+Land+Cover (Links to an external site.) Links to an external site.

You will need to use the SVC & LinearSVC classifiers from sklearn.svm. You will also need to use RandomForestClassifier (from HW #4) to compare performance of this model to support vector machine models.

1. Data Processing:

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

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

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"

d) Scale all features / predictors (NOT THE TARGET VARIABLE)

In [1]:

```
import numpy as np
import os
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [2]:

```
#a) import the data
train=pd.read_csv('train_data.csv')
test=pd.read_csv('test_data.csv')
#b) remove any rows that have missing data across both sets of data
test=test.dropna()
train=train.dropna()
#c) separate 'class' into y_train, y_test and remove from X_train, X_test
y_test=test['class']
y_train=train['class']
X_test=test.drop(['class'], axis=1)
X_train=train.drop(['class'], axis=1)
#d) scale all feature/predictors
scaler=StandardScaler()
X_train_scaled=scaler.fit_transform(X_train)
X_test_scaled=scaler.transform(X_test)
```

2. 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 [3]:

```
from sklearn.ensemble import RandomForestClassifier
forest_clf = RandomForestClassifier(random_state=1007)
rf_model=forest_clf.fit(X_train_scaled, y_train)
```

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

In [4]:

```
rf_predict=rf_model.predict(X_test_scaled)
```

c) Calculate the confusion matrix and classification report (both are in sklearn.metrics).

In [5]:

```
from sklearn.metrics import confusion_matrix, classification_report
print('Confusion Matrix\n')
print(confusion_matrix(y_test, rf_predict))
print('\nClassification Report\n')
print(classification_report(y_test, rf_predict))
```

```
Confusion Matrix

[[14  0  0  0  0  0  0  0  0]
 [ 2 20  0  3  0  0  0  0  0]
 [ 0  0 11  0  0  1  1  2  0]
 [ 0  4  0 18  0  0  0  1  0]
 [ 0  0  0  0 27  0  0  0  2]
 [ 1  0  1  0  0 13  0  0  0]
 [ 2  0  0  0  0  0 13  0  1]
 [ 0  2  0  5  1  0  0  6  0]
 [ 0  0  0  1  1  0  0  0 15]]

Classification Report

             precision    recall  f1-score   support

    asphalt       0.74      1.00      0.85        14
   building       0.77      0.80      0.78        25
        car       0.92      0.73      0.81        15
   concrete       0.67      0.78      0.72        23
      grass       0.93      0.93      0.93        29
       pool       0.93      0.87      0.90        15
     shadow       0.93      0.81      0.87        16
       soil       0.67      0.43      0.52        14
       tree       0.83      0.88      0.86        17

avg / total       0.82      0.82      0.81       168
```

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

In [6]:

```
rf_importances=rf_model.feature_importances_
features=X_train.columns
#get the indices of the importances, ordered by the underlying value, take top 5 (np.argsort goes
by ascending to go backwards from end)
indices = np.argsort(rf_importances)[-5:]
high_to_low=np.flip(indices, 0)
top_5_features=features[high_to_low]
top_5_imp=rf_importances[high_to_low]
top_5=pd.DataFrame({'Top 5 Features':top_5_features, 'Importance':top_5_imp})
top_5
```

Out[6]:

	Importance	Top 5 Features
0	0.064724	Mean_R
1	0.039948	NDVI_80
2	0.031781	NDVI_40
3	0.030223	Mean_NIR_100

4	Importance	Top 5 Features

3. 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 [7]:

```
from sklearn.svm import LinearSVC
lin_svm_clf = LinearSVC(random_state=49523)
lin_svm_model=lin_svm_clf.fit(X_train_scaled, y_train)
```

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

In [8]:

```
lin_svm_pred=lin_svm_model.predict(X_test_scaled)
```

c) Calculate the confusion matrix and classification report (both are in sklearn.metrics).

In [9]:

```
print('Confusion Matrix\n')
print(confusion_matrix(y_test, lin_svm_pred))
print('\nClassification Report\n')
print(classification_report(y_test, lin_svm_pred))
```

```
Confusion Matrix

[[13  0  0  0  0  0  1  0  0]
 [ 0 21  1  1  1  0  0  1  0]
 [ 0  2 12  0  0  0  0  0  1]
 [ 1  6  0 15  0  0  0  0  1]
 [ 0  0  0  1 26  0  0  0  2]
 [ 1  0  1  0  0 13  0  0  0]
 [ 2  0  0  0  0  0 14  0  0]
 [ 0  4  0  1  3  0  0  6  0]
 [ 0  0  0  1  6  0  0  0 10]]

Classification Report

             precision    recall  f1-score   support

   asphalt       0.76      0.93      0.84        14
  building       0.64      0.84      0.72        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       0.86      0.43      0.57        14
      tree       0.71      0.59      0.65        17

avg / total       0.79      0.77      0.77       168
```

4. Support Vector Machine Classifier + Linear Kernel + Grid Search:

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.

In [10]:

```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
C_range=np.arange(.01,10, .2)
param_grid={'C':C_range, 'kernel':['linear'], 'random_state':[1320]}
#gamma only relevant for 'rbf', 'poly' and 'sigmoid' so omitted above
C_range
```

Out[10]:

```
array([ 0.01,  0.21,  0.41,  0.61,  0.81,  1.01,  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,  4.21,  4.41,  4.61,  4.81,  5.01,  5.21,
        5.41,  5.61,  5.81,  6.01,  6.21,  6.41,  6.61,  6.81,  7.01,
        7.21,  7.41,  7.61,  7.81,  8.01,  8.21,  8.41,  8.61,  8.81,
        9.01,  9.21,  9.41,  9.61,  9.81])
```

In [11]:

```
svc=SVC()
grid_SVC=GridSearchCV(svc, param_grid, cv = 5, refit = True, n_jobs=-1, verbose = 5)
grid_SVC_model=grid_SVC.fit(X_train_scaled, y_train)
```

```
Fitting 5 folds for each of 50 candidates, totalling 250 fits
```

```
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    2.5s
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:    9.3s
[Parallel(n_jobs=-1)]: Done 250 out of 250 | elapsed:   10.8s finished
```

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 [12]:

```
print('Best Performing Parameters\n')
print(grid_SVC_model.best_params_)
print('\nBest Performing Model\n')
print(grid_SVC_model.best_estimator_)
```

```
Best Performing Parameters

{'C': 0.01, 'kernel': 'linear', 'random_state': 1320}

Best Performing Model

SVC(C=0.01, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=1320, shrinking=True,
  tol=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 [13]:

```
best_grid_SVC_model=grid_SVC_model.best_estimator_
best_grid_SVC_model_pred=best_grid_SVC_model.predict(X_test_scaled)
```

d) Calculate the confusion matrix and classification report (both are in sklearn.metrics).

In [14]:

```
print('Confusion Matrix\n')
print(confusion_matrix(y_test, best_grid_SVC_model_pred))
print('\nClassification Report\n')
print(classification_report(y_test, best_grid_SVC_model_pred))
```

```
Confusion Matrix

[[13  0  0  0  0  0  1  0  0]
 [ 0 22  0  2  1  0  0  0  0]
 [ 0  1 14  0  0  0  0  0  0]
 [ 0  5  0 17  0  0  0  1  0]
 [ 0  0  0  1 25  0  0  0  3]
 [ 0  0  0  0  0 14  1  0  0]
 [ 1  0  0  0  0  0 15  0  0]
 [ 0  3  0  5  2  0  0  4  0]
 [ 0  0  0  1  2  0  0  0 14]]

Classification Report

             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

avg / total       0.83      0.82      0.81       168
```

5. 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 [15]:

```
#because it is poly you can tune gamma
param_grid2={'C':C_range, 'kernel':['poly'], 'degree':[2,3,4,5,6], 'gamma':[0.1, .5, 1, 2, 5, 10], '
random_state':[9876]}
svc_poly=SVC()
grid_SVC_poly=GridSearchCV(svc_poly, param_grid2, cv = 5, refit = True, n_jobs=-1, verbose = 5)
grid_SVC_poly_model=grid_SVC_poly.fit(X_train_scaled, y_train)
```

```
Fitting 5 folds for each of 1500 candidates, totalling 7500 fits

[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    2.6s
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:    8.9s
[Parallel(n_jobs=-1)]: Done 233 tasks      | elapsed:   11.1s
[Parallel(n_jobs=-1)]: Done 485 tasks      | elapsed:   14.0s
[Parallel(n_jobs=-1)]: Done 809 tasks      | elapsed:   17.8s
[Parallel(n_jobs=-1)]: Done 1205 tasks      | elapsed:   22.4s
[Parallel(n_jobs=-1)]: Done 1673 tasks      | elapsed:   28.0s
[Parallel(n_jobs=-1)]: Done 2213 tasks      | elapsed:   34.5s
[Parallel(n_jobs=-1)]: Done 2825 tasks      | elapsed:   41.7s
[Parallel(n_jobs=-1)]: Done 3509 tasks      | elapsed:   49.7s
[Parallel(n_jobs=-1)]: Done 4265 tasks      | elapsed:   59.2s
[Parallel(n_jobs=-1)]: Done 5093 tasks      | elapsed:  1.1min
```

```
[Parallel(n_jobs=-1)]: Done 5993 tasks      | elapsed:  1.3min
[Parallel(n_jobs=-1)]: Done 6965 tasks      | elapsed:  1.5min
[Parallel(n_jobs=-1)]: Done 7485 out of 7500 | elapsed:  1.6min remaining:    0.1s
[Parallel(n_jobs=-1)]: Done 7500 out of 7500 | elapsed:  1.6min finished
```

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 [16]:

```
print('Best Performing Parameters\n')
print(grid_SVC_poly_model.best_params_)
print('\nBest Performing Model\n')
print(grid_SVC_poly_model.best_estimator_)
```

```
Best Performing Parameters

{'C': 0.01, 'degree': 3, 'gamma': 0.1, 'kernel': 'poly', 'random_state': 9876}

Best Performing Model

SVC(C=0.01, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.1, kernel='poly',
  max_iter=-1, probability=False, random_state=9876, shrinking=True,
  tol=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 [17]:

```
best_grid_SVC_poly_model=grid_SVC_poly.best_estimator_
best_grid_SVC_poly_model_pred=best_grid_SVC_poly_model.predict(X_test_scaled)
```

d) Calculate the confusion matrix and classification report (both are in sklearn.metrics).

In [18]:

```
print('Confusion Matrix\n')
print(confusion_matrix(y_test, best_grid_SVC_poly_model_pred))
print('\nClassification Report\n')
print(classification_report(y_test, best_grid_SVC_poly_model_pred))
```

```
Confusion Matrix

[[13  0  0  0  0  0  1  0  0]
 [ 0 22  0  2  1  0  0  0  0]
 [ 0  2 11  0  0  1  0  1  0]
 [ 0  3  0 19  0  0  0  1  0]
 [ 0  0  0  0 26  0  0  1  2]
 [ 0  0  0  0  0 14  1  0  0]
 [ 2  0  0  0  0  0 14  0  0]
 [ 0  1  0  4  4  0  0  5  0]
 [ 0  0  0  1  3  0  0  0 13]]

Classification Report

              precision    recall  f1-score   support

     asphalt       0.87      0.93      0.90        14
    building       0.79      0.88      0.83        25
         car       1.00      0.73      0.85        15
    concrete       0.73      0.83      0.78        23
       grass       0.76      0.90      0.83        29
        pool       0.93      0.93      0.93        15
      shadow       0.88      0.88      0.88        16
        soil       0.62      0.36      0.45        14
        tree       0.87      0.76      0.81        17

avg / total       0.82      0.82      0.81       168
```

```
avg / total       0.82      0.82      0.81       188
```

6. 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: 0.1, 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 [19]:

```
param_grid3={'C':C_range, 'kernel':['rbf'], 'degree':[2,3,4,5,6], 'gamma':[0.1, 1, 10, 100],
'random_state':[4321]}
svc_rbf=SVC()
grid_SVC_rbf=GridSearchCV(svc_rbf, param_grid3, cv = 5, refit = True, n_jobs=-1, verbose = 5)
grid_SVC_rbf_model=grid_SVC_rbf.fit(X_train_scaled, y_train)
```

```
Fitting 5 folds for each of 1000 candidates, totalling 5000 fits
```

```
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    2.5s
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:    9.3s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:   10.9s
[Parallel(n_jobs=-1)]: Done 272 tasks      | elapsed:   14.8s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:   18.7s
[Parallel(n_jobs=-1)]: Done 632 tasks      | elapsed:   23.1s
[Parallel(n_jobs=-1)]: Done 866 tasks      | elapsed:   28.5s
[Parallel(n_jobs=-1)]: Done 1136 tasks      | elapsed:   34.2s
[Parallel(n_jobs=-1)]: Done 1442 tasks      | elapsed:   41.9s
[Parallel(n_jobs=-1)]: Done 1784 tasks      | elapsed:   49.2s
[Parallel(n_jobs=-1)]: Done 2162 tasks      | elapsed:   56.8s
[Parallel(n_jobs=-1)]: Done 2576 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done 3026 tasks      | elapsed:  1.3min
[Parallel(n_jobs=-1)]: Done 3512 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done 4034 tasks      | elapsed:  1.6min
[Parallel(n_jobs=-1)]: Done 4592 tasks      | elapsed:  1.8min
[Parallel(n_jobs=-1)]: Done 5000 out of 5000 | elapsed:  2.0min finished
```

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]:

```
print('Best Performing Parameters\n')
print(grid_SVC_rbf_model.best_params_)
print('\nBest Performing Model\n')
print(grid_SVC_rbf_model.best_estimator_)
```

```
Best Performing Parameters

{'C': 1.4100000000000001, 'degree': 2, 'gamma': 0.1, 'kernel': 'rbf', 'random_state': 4321}

Best Performing Model

SVC(C=1.4100000000000001, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=2, gamma=0.1, kernel='rbf',
  max_iter=-1, probability=False, random_state=4321, shrinking=True,
  tol=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]:

```
best_grid_SVC_rbf_model=grid_SVC_rbf_model.best_estimator_
best_grid_SVC_rbf_pred=best_grid_SVC_rbf_model.predict(X_test_scaled)
```

d) Calculate the confusion matrix and classification report (both are in sklearn.metrics).

In [22]:

```
print('Confusion Matrix\n')
print(confusion_matrix(y_test, best_grid_SVC_rbf_pred))
print('\nClassification Report\n')
print(classification_report(y_test, best_grid_SVC_rbf_pred))
```

```
Confusion Matrix

[[ 0  0  0 14  0  0  0  0  0]
 [ 0 17  0  7  0  0  0  0  1]
 [ 0  0  0 15  0  0  0  0  0]
 [ 0  3  0 20  0  0  0  0  0]
 [ 0  0  0  9  9  0  0  0 11]
 [ 0  0  0 15  0  0  0  0  0]
 [ 0  0  0 12  0  0  0  0  4]
 [ 0  0  0 13  1  0  0  0  0]
 [ 0  0  0  1  0  0  0  0 16]]

Classification Report

             precision    recall  f1-score   support

    asphalt       0.00      0.00      0.00        14
   building       0.85      0.68      0.76        25
        car       0.00      0.00      0.00        15
   concrete       0.19      0.87      0.31        23
      grass       0.90      0.31      0.46        29
       pool       0.00      0.00      0.00        15
     shadow       0.00      0.00      0.00        16
       soil       0.00      0.00      0.00        14
       tree       0.50      0.94      0.65        17

avg / total       0.36      0.37      0.30       168
```

```
C:\Users\mjdun\Anaconda\lib\site-packages\sklearn\metrics\classification.py:1135:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with
no predicted samples.
  'precision', 'predicted', average, warn_for)
```

7. 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.

Based on the Classification Report, there was little to no difference between: (1) the Baseline Random Forest (F1 Score=0.81), (2) the Linear SVC w/o Grid Search (F1 Score=0.77), (3) the Linear SVC w/ Grid Search (F1 Score=0.81), and (4) the SVC w/ Polynomial Kernel and Grid Search (F1 Score=0.81). The SVC with RBF Kernel performed decidedly worse (F1 Score=0.38).

In choosing a best model we would look at the Baseline Random Forest, the Linear SVC w/ Grid Search, and the SVC w/ Polynomial Kernel and Grid Search. The Linear SVC w/ Grid Search technically has the highest precision (0.83 vs 0.82) and it would be the easiest to interpret and explain. I suspect that the Polynomial Kernel is not really appropriate for the data and that with some tuning (i.e. the minimum possible values for C and gamma) it happens to approach the accuracy of the other models. Also, it seems that the precision of the Linear SVC model is more reasonably distributed with fewer extremes than the other models. It has only one precision score in the 0.70 to 0.79 range whereas the Polynomial Kernel has three.

For these reasons I would say the Linear SVC with Grid Search performs the best.

8. 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?

These models used the linear, polynomial, and rbf kernels respectively. The benefits of using the polynomial or rbf kernel is that they can transform the data such that a linear decision boundary can work reasonably well where it did not before. The problem is that often one cannot know whether they will be an improvement over the standard linear model until the data is transformed, and there is no guarantee that it will be an improvement. These kernel sometimes make the model much better by transforming the data and sometimes much worse, as we see with the rbf kernel here. Furthermore these kernels can be computationally expensive and be difficult to interpret.

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

The C parameter of the SVC function in Python is the penalty for the error term. Its default value = 1. As it becomes smaller the model become MORE tolerant of individual errors in classifying data points, and as it become larger the model is LESS tolerant of individual errors.

This is tuned to increase a classifier's generalizability (performance on test data). Generally one cannot draw a clean line between two classes of data. Some points will be within the Classifier Margin (the distance between the decision boundary and the first data point in a class) or misclassified (on the wrong side of the decision boundaryaltogether). How tolerant the model is of these when fitting on the training data will influence not only the width of the margin but also where the decision boundary is drawn (because it is influenced by the maximum width one can have before hitting a data point).

The reason we tune the C parameter when fitting the model is to increase GENERALIZABILITY. Minimizing error on training data at the cost of increasing error is overfitting. If we are intolerant of classification errors on our training data (i.e. increase our C) at the expense of making classification errors on our test data we are overfitting.

10. It is also important to check for overfitting: For your best performing model provide metrics for the training and test sets and explain whether your model is overfitting the data.

The accuracy_score on our original trainings set (scaled) for the Linear SVC w/ Grid Search is:

In [23]:

```
from sklearn.metrics import accuracy_score
best_model_train_pred=best_grid_SVC_model.predict(X_train_scaled)
accuracy_score(y_train, best_model_train_pred)
```

Out[23]:

```
0.8875739644970414
```

And the accuracy_score of our test set (scaled) is:

In [24]:

```
best_model_test_pred=best_grid_SVC_model.predict(X_test_scaled)
accuracy_score(y_test, best_model_test_pred)
```

Out[24]:

```
0.8214285714285714
```

We see a not insignificant drop off in accuracy from train to test. That is a sign of overfitting. The solution for this is to lower our C parameter even more. It is already at the lowest value allowed in the Grid Search. This was the only parameter we tuned under the Linear SVC model. Other parameters, e.g. gamma are not applicable to this model.