• Linear SVM Implementation Steps:

① Data Pre-processing –

```
df = pd.read_csv('___')
x = df.drop('target', axis=1)
y = df['target']
train_test_split
```

↓

② Create & Train SVM Model –

```
from sklearn.svm import SVC
svm_clf = SVC(kernel = 'linear', C=1.0)
svm_clf.fit(x_train, y_train)
```

↓

③ Predict Test Set Results –

```
y_pred = svm_clf.predict(x_test)
```

↓

④ Evaluate Model Performance – sklearn.metrics

accuracy_score, classification_report, confusion_matrix

↓

⑤ Visualize the Results –

sklearn.inspection ⇒ DecisionBoundaryDisplay

```
DBD.from_estimator( svm_clf, x_train,
        response_method = 'predict',
        cmap = plt.cm.spectral, alpha = 0.8,
        xlabel = '___',
        ylabel = '___')

plt.scatter (x_train [x], x_train [y],
        c = y_train,
        s = 20, edgecolors = 'k')

plt.show()
```

10

2) Non-Linear SVM :-

- Handle both linear & non-linear data.
- Well handle high-dim feature space & complex data.
- Classification : Find hyperplane with greatest margin betⁿ classes.
- Regression: Fit a linear function predict continuous target variables.

> Non-Linear SVM necessary, when data not linearly separable in original feature space. Non-Lin SVM utilize kernel funⁿ to map data into higher-dim space, where linear separation become possible.

- Kernel - Math funⁿ used in SVM to map original data points into high-dim feature spaces, where data can linearly separable.
  - Funⁿ: Radial basis function (rbf), linear, polynomial & sigmoid.

- Non-Linear Decision Boundaries : Non-linear SVM allow create complex decision boundaries can accurately separate data points of diff. classes.
  - By transforming data using kernel funⁿ, SVM can capture non-linear relationships betⁿ features.

- Regularization Parameter (c) : Control trade-off betⁿ misclassification of training examples & margin width.
  - Help prevent overfitting & influence model's complexity.

Non-Linear SVM Implementation Steps:

(1) Data Pre-processing –

```
df = pd.read_csv('___')
x = df.drop('target', axis=1).values
y = df['target']
Train-test-split
```

Extract data as np.ndarray not Data Frame

(2) Create & Train SVM Model –

```
from sklearn.svm import SVC
svm_clf = SVC(kernel='poly', degree = 5,
              random_state= 42)
svm_clf.fit(x_train, y_train)
```

(3) Predict Test Set Results –

```
y_pred = svm_clf.predict(x_test)
```

(4) Evaluate Model Performance –

```
from sklearn.metrics
accuracy_score(y_test, y_pred)
classification_report(y_test, y_pred)
confusion_matrix(y_test, y_pred)
```

(5) Visualize the Results :

#1: Define Grid for visualization -

```
x_min = x[:, 0].min() - 1
x_max = x[:, 0].max() + 1
y_min = x[:, 1].min() - 1
y_max = x[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                     np.arange(y_min, y_max, 0.02))
```

↓

#2: Make Predictions on Mesh Grid -

```
Z = svm_clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
```

↓

#3: Plot Decision Boundary & Data Points -

```
plt.figure(figsize = (8,6))
plt.contourf(xx, yy, Z, alpha = 0.8)
plt.scatter(x[:, 0], x[:, 1], c = y,
            edgecolors = 'k', marker = 'o')
plt.xlabel('first_feature')
plt.ylabel('second_feature')
plt.title('______')
plt.show()
```