# VIT UNIVERSITY, ANDHRA PRADESH
## School of CSE
## CSE3008 - Introduction to Machine Learning
## Lab Experiment-8
## ( **Support vector machine** )

Faculty-**Dr. B. SRINIVASA RAO**

**Name-**Neeraj Guntuku
**R.No-**18MIS7071
**Slot-**L55+L56

Date- 3 April 2021

---

Support vector machine

### *Support vector machine*

```
[1] import numpy as np
    import cvxopt
    from sklearn.datasets.samples_generator import make_blobs
    from sklearn.model_selection import train_test_split
    from matplotlib import pyplot as plt
    from sklearn.svm import LinearSVC
    from sklearn.metrics import confusion_matrix
```

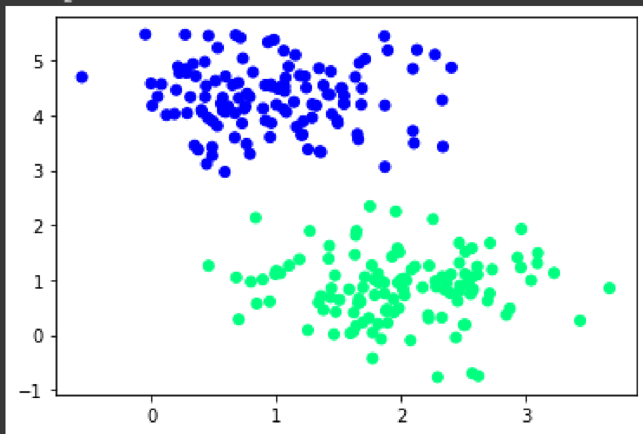
```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:144: FutureW
  warnings.warn(message, FutureWarning)
```

```
[2]
class SVM:
  def fit(self, X, y):
        n_samples, n_features = X.shape# P = X^T X
        K = np.zeros((n_samples, n_samples))
        for i in range(n_samples):
            for j in range(n_samples):
                K[i,j] = np.dot(X[i], X[j])
        P = cvxopt.matrix(np.outer(y, y) * K)# q = -1 (1xN)
        q = cvxopt.matrix(np.ones(n_samples) * -1)# A = y^T
        A = cvxopt.matrix(y, (1, n_samples))# b = 0
        b = cvxopt.matrix(0.0)# -1 (NxN)
        G = cvxopt.matrix(np.diag(np.ones(n_samples) * -1))# 0 (1xN)
        h = cvxopt.matrix(np.zeros(n_samples))
        solution = cvxopt.solvers.qp(P, q, G, h, A, b)# Lagrange multipliers
        a = np.ravel(solution['x'])# Lagrange have non zero lagrange multipliers
        sv = a > 1e-5
        ind = np.arange(len(a))[sv]
        self.a = a[sv]
        self.sv = X[sv]
        self.sv_y = y[sv]# Intercept
        self.b = 0
        for n in range(len(self.a)):
            self.b += self.sv_y[n]
            self.b -= np.sum(self.a * self.sv_y * K[ind[n], sv])
        self.b /= len(self.a)# Weights
        self.w = np.zeros(n_features)
        for n in range(len(self.a)):
            self.w += self.a[n] * self.sv_y[n] * self.sv[n]

  def project(self, X):
    return np.dot(X, self.w) + self.b


  def predict(self, X):
    return np.sign(self.project(X))
```

```
[3] X, y = make_blobs(n_samples=250, centers=2,random_state=0, cluster_std=0.60)
    y[y == 0] = -1
    tmp = np.ones(len(X))
    y = tmp * y
```

```
[4] plt.scatter(X[:, 0], X[:, 1], c=y, cmap='winter')
```

```
<matplotlib.collections.PathCollection at 0x7f6a4a4e83d0>
```



```
[5] X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

```
[6] svm = SVM()
    svm.fit(X_train, y_train)
```

```
     pcost       dcost       gap    pres   dres
 0: -1.8226e+01 -3.4458e+01  6e+02  2e+01  2e+00
 1: -2.5252e+01 -1.8773e+01  2e+02  9e+00  7e-01
 2: -5.3459e+01 -3.2711e+01  2e+02  7e+00  6e-01
 3: -7.8360e+01 -2.6482e+01  1e+02  4e+00  3e-01
 4: -5.6818e+00 -5.1750e+00  1e+01  2e-01  1e-02
 5: -3.6906e+00 -4.1082e+00  4e-01  4e-16  9e-15
 6: -4.0061e+00 -4.0104e+00  4e-03  1e-15  6e-15
 7: -4.0094e+00 -4.0094e+00  4e-05  1e-15  4e-15
 8: -4.0094e+00 -4.0094e+00  4e-07  2e-15  7e-15
Optimal solution found.
```

```
[7] def f(x, w, b, c=0):
      return (-w[0] * x - b + c) / w[1]
      plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='winter')# w.x + b = 0
      a0 = -4; a1 = f(a0, svm.w, svm.b)
      b0 = 4; b1 = f(b0, svm.w, svm.b)
      plt.plot([a0,b0], [a1,b1], 'k')# w.x + b = 1
      a0 = -4; a1 = f(a0, svm.w, svm.b, 1)
      b0 = 4; b1 = f(b0, svm.w, svm.b, 1)
      plt.plot([a0,b0], [a1,b1], 'k--')# w.x + b = -1
      a0 = -4; a1 = f(a0, svm.w, svm.b, -1)
      b0 = 4; b1 = f(b0, svm.w, svm.b, -1)
      plt.plot([a0,b0], [a1,b1], 'k--')
```

```
[8] y_pred = svm.predict(X_test)
    confusion_matrix(y_test, y_pred)
```

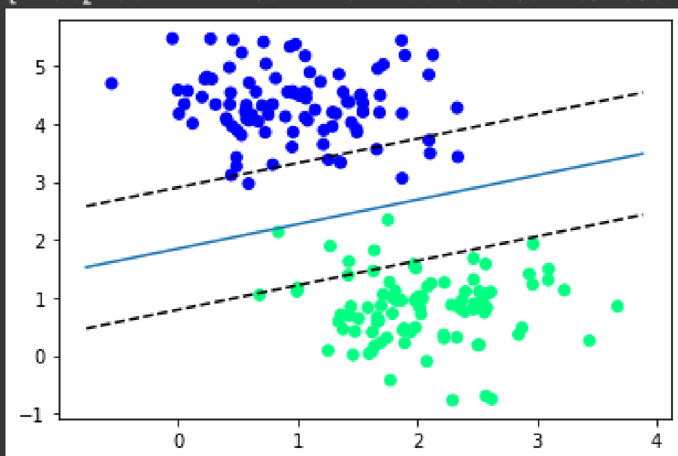
```
array([[29,  0],
       [ 0, 34]])
```

```
[9] svc = LinearSVC()
    svc.fit(X_train, y_train)
```

```
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=None, tol=0.0001,
          verbose=0)
```

```
[10] plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='winter');
     ax = plt.gca()
     xlim = ax.get_xlim()
     w = svc.coef_[0]
     a = -w[0] / w[1]
     xx = np.linspace(xlim[0], xlim[1])
     yy = a * xx - svc.intercept_[0] / w[1]
     plt.plot(xx, yy)
     yy = a * xx - (svc.intercept_[0] - 1) / w[1]
     plt.plot(xx, yy, 'k--')
     yy = a * xx - (svc.intercept_[0] + 1) / w[1]
     plt.plot(xx, yy, 'k--')
```

```
[<matplotlib.lines.Line2D at 0x7f6a41b9fe50>]
```




```
[11] y_pred = svc.predict(X_test)
     confusion_matrix(y_test, y_pred)
```

```
array([[29,  0],
       [ 0, 34]])
```

# Support vector machine (Radial Basis Function (RBF) kernel )

## Radial Basis Function (RBF) kernel

```
[1] import pandas as pd # for data manipulation
    import numpy as np # for data manipulation

    from sklearn.model_selection import train_test_split # for splitting the data into train and test samples
    from sklearn.metrics import classification_report # for model evaluation metrics
    from sklearn.svm import SVC # for Support Vector Classification model

    import plotly.express as px  # for data visualization
    import plotly.graph_objects as go # for data visualization
```

```
[2] # Read in the csv
    df=pd.read_csv('games.csv', encoding='utf-8')

    # Difference between white rating and black rating - independent variable
    df['rating_difference']=df['white_rating']-df['black_rating']

    # White wins flag (1=win vs. 0=not-win) - dependent (target) variable
    df['white_win']=df['winner'].apply(lambda x: 1 if x=='white' else 0)

    # Print a snapshot of a few columns
    df.iloc[:,[0,1,5,6,8,9,10,11,13,16,17]]
```

| | id | rated | victory_status | winner | white_id | white_rating | black_id | black_rating | opening_eco | rati |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | TZJHLljE | False | outoftime | white | bourgris | 1500 | a-00 | 1191 | D10 | |
| 1 | l1NXvwaE | True | resign | black | a-00 | 1322 | skinnerua | 1261 | B00 | |
| 2 | mIlCvQHh | True | mate | white | ischia | 1496 | a-00 | 1500 | C20 | |
| 3 | kWKvrqYL | True | mate | white | daniamurashov | 1439 | adivanov2009 | 1454 | D02 | |
| 4 | 9tXo1AUZ | True | mate | white | nik221107 | 1523 | adivanov2009 | 1469 | C41 | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| 20053 | EfqH7VVH | True | resign | white | belcolt | 1691 | jamboger | 1220 | A80 | |
| 20054 | WSJDhbPl | True | mate | black | jamboger | 1233 | farrukhasomiddinov | 1196 | A41 | |
| 20055 | yrAas0Kj | True | mate | white | jamboger | 1219 | schaaksmurf3 | 1286 | D00 | |
| 20056 | b0v4tRyF | True | resign | white | marcodisogno | 1360 | jamboger | 1227 | B07 | |
| 20057 | N8G2JHGG | True | mate | black | jamboger | 1235 | ffbob | 1339 | D00 | |

```python
[3] def fitting(X, y, C, gamma):
        # Create training and testing samples
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

        # Fit the model
        # Note, available kernels: {'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'}, default='rbf'
        model = SVC(kernel='rbf', probability=True, C=C, gamma=gamma)
        clf = model.fit(X_train, y_train)

        # Predict class labels on training data
        pred_labels_tr = model.predict(X_train)
        # Predict class labels on a test data
        pred_labels_te = model.predict(X_test)

        # Use score method to get accuracy of the model
        print('----- Evaluation on Test Data -----')
        score_te = model.score(X_test, y_test)
        print('Accuracy Score: ', score_te)
        # Look at classification report to evaluate the model
        print(classification_report(y_test, pred_labels_te))
        print('--------------------------------------------------------')

        print('----- Evaluation on Training Data -----')
        score_tr = model.score(X_train, y_train)
        print('Accuracy Score: ', score_tr)
        # Look at classification report to evaluate the model
        print(classification_report(y_train, pred_labels_tr))
        print('--------------------------------------------------------')

        # Return relevant data for chart plotting
        return X_train, X_test, y_train, y_test, clf
```

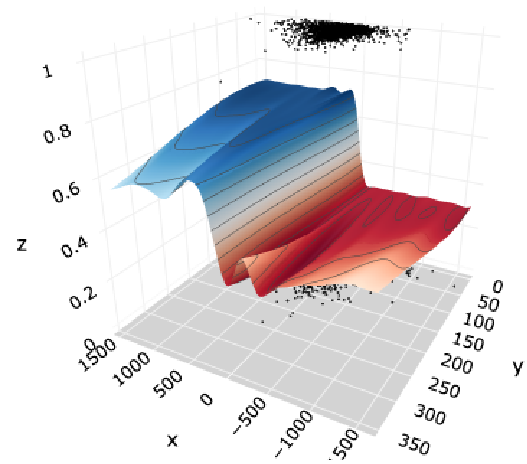
```python
[4] def Plot_3D(X, X_test, y_test, clf):

        # Specify a size of the mesh to be used
        mesh_size = 5
        margin = 1

        # Create a mesh grid on which we will run our model
        x_min, x_max = X.iloc[:, 0].fillna(X.mean()).min() - margin, X.iloc[:, 0].fillna(X.mean()).max() + margin
        y_min, y_max = X.iloc[:, 1].fillna(X.mean()).min() - margin, X.iloc[:, 1].fillna(X.mean()).max() + margin
        xrange = np.arange(x_min, x_max, mesh_size)
        yrange = np.arange(y_min, y_max, mesh_size)
        xx, yy = np.meshgrid(xrange, yrange)

        # Calculate predictions on grid
        Z = clf.predict_proba(np.c_[xx.ravel(), yy.ravel()])[:, 1]
        Z = Z.reshape(xx.shape)

        # Create a 3D scatter plot with predictions
        fig = px.scatter_3d(x=X_test['rating_difference'], y=X_test['turns'], z=y_test,
                         opacity=0.8, color_discrete_sequence=['black'])

        # Set figure title and colors
        fig.update_layout(#title_text="Scatter 3D Plot with SVM Prediction Surface",
                          paper_bgcolor = 'white',
                          scene = dict(xaxis=dict(backgroundcolor='white',
                                                  color='black',
                                                  gridcolor='#f0f0f0'),
                                       yaxis=dict(backgroundcolor='white',
                                                  color='black',
                                                  gridcolor='#f0f0f0'
                                                  ),
                                       zaxis=dict(backgroundcolor='lightgrey',
```

```python
    # Set figure title and colors
    fig.update_layout(#title_text="Scatter 3D Plot with SVM Prediction Surface",
                      paper_bgcolor = 'white',
                      scene = dict(xaxis=dict(backgroundcolor='white',
                                              color='black',
                                              gridcolor='#f0f0f0'),
                                   yaxis=dict(backgroundcolor='white',
                                              color='black',
                                              gridcolor='#f0f0f0'
                                              ),
                                   zaxis=dict(backgroundcolor='lightgrey',
                                              color='black',
                                              gridcolor='#f0f0f0',
                                              )))
    # Update marker size
    fig.update_traces(marker=dict(size=1))

    # Add prediction plane
    fig.add_traces(go.Surface(x=xrange, y=yrange, z=Z, name='SVM Prediction',
                              colorscale='RdBu', showscale=False,
                              contours = {"z": {"show": True, "start": 0.2, "end": 0.8, "size": 0.05}}))
    fig.show()
```

```python
[5] # Select data for modeling
    X=df[['rating_difference', 'turns']]
    y=df['white_win'].values

    # Fit the model and display results
    X_train, X_test, y_train, y_test, clf = fitting(X, y, 1, 'scale')
```

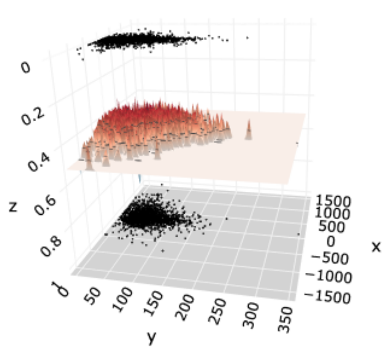
```
----- Evaluation on Test Data -----
Accuracy Score:  0.6530408773678963
              precision    recall  f1-score   support

           0       0.64      0.70      0.67      2024
           1       0.66      0.60      0.63      1988

    accuracy                           0.65      4012
   macro avg       0.65      0.65      0.65      4012
weighted avg       0.65      0.65      0.65      4012

-------------------------------------------------------
----- Evaluation on Training Data -----
Accuracy Score:  0.6468901907017325
              precision    recall  f1-score   support

           0       0.64      0.68      0.66      8033
           1       0.66      0.62      0.64      8013

    accuracy                           0.65     16046
   macro avg       0.65      0.65      0.65     16046
weighted avg       0.65      0.65      0.65     16046

-------------------------------------------------------
```

```
[6] Plot_3D(X, X_test, y_test, clf)
```



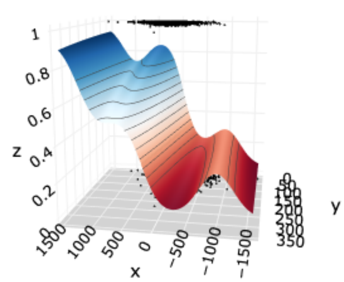
```
[7] # Select data for modeling
    X=df[['rating_difference', 'turns']]
    y=df['white_win'].values

    # Fit the model and display results
    X_train, X_test, y_train, y_test, clf = fitting(X, y, 1, 0.1)

    # Plot 3D chart
    Plot_3D(X, X_test, y_test, clf)
```

```
----- Evaluation on Test Data -----
Accuracy Score:  0.603938185443669
              precision    recall  f1-score   support

           0       0.60      0.64      0.62      2024
           1       0.61      0.57      0.59      1988

    accuracy                           0.60      4012
   macro avg       0.60      0.60      0.60      4012
weighted avg       0.60      0.60      0.60      4012

--------------------------------------------------------
----- Evaluation on Training Data -----
Accuracy Score:  0.8003240683036271
              precision    recall  f1-score   support

           0       0.80      0.81      0.80      8033
           1       0.80      0.80      0.80      8013

    accuracy                           0.80     16046
   macro avg       0.80      0.80      0.80     16046
weighted avg       0.80      0.80      0.80     16046

--------------------------------------------------------
```



```
[8] # Select data for modeling
    X=df[['rating_difference', 'turns']]
    y=df['white_win'].values

    # Fit the model and display results
    X_train, X_test, y_train, y_test, clf = fitting(X, y, 1, 0.000001)

    # Plot 3D chart
    Plot_3D(X, X_test, y_test, clf)
```

```
----- Evaluation on Test Data -----
Accuracy Score:  0.6602691924227319
              precision    recall  f1-score   support

           0       0.65      0.70      0.68      2024
           1       0.67      0.62      0.64      1988

    accuracy                           0.66      4012
   macro avg       0.66      0.66      0.66      4012
weighted avg       0.66      0.66      0.66      4012

--------------------------------------------------------
----- Evaluation on Training Data -----
Accuracy Score:  0.6463916240807678
              precision    recall  f1-score   support

           0       0.64      0.67      0.65      8033
           1       0.65      0.62      0.64      8013

    accuracy                           0.65     16046
   macro avg       0.65      0.65      0.65     16046
weighted avg       0.65      0.65      0.65     16046

--------------------------------------------------------
```




***