Unit 3
Support Vector Classification

LINEAR SVC

Demo Code:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import datasets
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split

data=pd.read_csv("Social_Network_Ads.csv")
print(data)
print(head())
data.shape

x=data.iloc[:,[2,3]]    # Age and Salary as an input variable
y=data.iloc[:,4]        # Purchased as an output variable
x.head()   # Print input data
y.head()   # Print Output data

from.sklearn.model.selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(x,y,test_size=0.20,rand
om_state=0)   # 20% as an testing data and remaining for training
```

```
print(X_train)
X_train.shape
X_test.shape
Y_train.shape
Y_test.shape
# Scale the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train the classifier
classifier = SVC(kernel='linear', random_state=0)
classifier.fit(X_train, Y_train)

# Predicting the test set results
Y_pred = classifier.predict(X_test)
print(Y_pred)

from sklearn import metrics
print("Accuracy Score: With linear kernel")
print(metrics.accuracy_score(Y_test,Y_pred))

# Plotting the test set results
plt.scatter(X_test[:, 0], X_test[:, 1], c=Y_test)

# Create a hyperplane
#calculation of the hyperplane equation
w = classifier.coef_[0]
a = -w[0] / w[1]
```

```
xx = np.linspace(X_test[:, 0].min(), X_test[:, 0].max())
yy = a * xx - (classifier.intercept_[0] / w[1])

# Plot hyperplane
plt.plot(xx, yy)
plt.axis("off")
plt.show()
```

#For reference
https://www.youtube.com/watch?v=Gl7sqGeblfw

NON-LINEAR SVC

Non-Linear SVM extends SVM to handle complex, non-linearly separable data using kernels. Kernels enable SVM to work in higher dimensions where data can become linearly separable.

Nonlinear SVM was introduced when the data cannot be separated by a linear decision boundary in the original feature space. The kernel function computes the similarity between data points allowing SVM to capture complex patterns and nonlinear relationships between features. This enables nonlinear SVM to form curved or circular decision boundaries with the help of kernels.

Popular kernel functions in SVM

- **Radial Basis Function (RBF)**: Captures patterns in data by measuring the distance between points and is ideal for circular or spherical relationships.

- **Linear Kernel**: Works for data that is a linearly separable problem without complex transformations.
- **Polynomial Kernel**: Models more complex relationships using polynomial equations.
- **Sigmoid Kernel**: Mimics neural network behavior using sigmoid function and is suitable for specific non-linear problems.

DEMO CODE:

Support Vector Classification

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn import metrics

# Load the dataset
data = pd.read_csv("Social_Network_Ads.csv")
print(data.head())  # Corrected from head() to data.head()
print(data.shape)

# Prepare input (X) and output (Y) variables
x = data.iloc[:, [2, 3]]  # Age and Salary as input variables
y = data.iloc[:, 4]       # Purchased as output variable
```

```
# Split the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size=0.20,
random_state=0)  # 20% test data

# Print shapes of training and testing data
print(X_train.shape, X_test.shape, Y_train.shape, Y_test.shape)

# Scale the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train the classifier with a polynomial kernel
classifier = SVC(kernel='poly', degree=4)
classifier.fit(X_train, Y_train)

# Predict the test set results
Y_pred = classifier.predict(X_test)
print(Y_pred)

# Calculate and print accuracy score
print("Accuracy Score: With polynomial kernel")
print(metrics.accuracy_score(Y_test, Y_pred))

# Visualize the decision boundary using a contour plot
x_min, x_max = X_test[:, 0].min() - 1, X_test[:, 0].max() + 1
y_min, y_max = X_test[:, 1].min() - 1, X_test[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                     np.arange(y_min, y_max, 0.1))
```

```
# Predict on the grid
Z = classifier.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot decision boundary and the test data points
plt.contourf(xx, yy, Z, alpha=0.75, cmap=plt.cm.Paired)
plt.scatter(X_test[:, 0], X_test[:, 1], c=Y_test, edgecolors='k',
marker='o', cmap=plt.cm.Paired)
plt.title("SVM with Polynomial Kernel")
plt.xlabel("Age")
plt.ylabel("Salary")
plt.show()
```

1. **Set the boundaries for the plot:**

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

X_test is the test dataset, where each point has two features (so X_test[:, 0] is the first feature and X_test[:, 1] is the second feature). The above lines calculate the minimum and maximum values of the test data points in both feature dimensions (x and y) and extend the

`range by 1 unit on each side, creating a margin around the data.`

2. Generate a mesh grid for contour plotting:

xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
np.arange(y_min, y_max, 0.1))

`np.meshgrid` creates a grid of points in the feature space. The `np.arange` function generates values for `xx` (x-axis) and `yy` (y-axis) within the defined boundaries.

The step size of `0.1` means the grid will have a high resolution, and the classifier will make predictions for many points.

3. **Make predictions on the grid:**

Z = classifier.predict(np.c_[xx.ravel(), yy.ravel()])

`np.c_[xx.ravel(), yy.ravel()]` stacks the `xx` and `yy` grids together, creating a list of coordinates to predict for each point in the mesh grid.

`classifier.predict()` is used to make predictions for each of these grid points. This results in an array `Z` of predicted class labels for each point.

4. Reshape the predictions back to the grid shape:

Z = Z.reshape(xx.shape)

`Z.reshape(xx.shape)` **reshapes the array** `Z` **back into the shape of the mesh grid so that each grid point has its corresponding predicted class label.**

5. Plot the decision boundary:
plt.contourf(xx, yy, Z, alpha=0.75, cmap=plt.cm.Paired)

- `plt.contourf()` **creates a filled contour plot. It plots the decision boundaries (the regions where the classifier predicts different classes) by filling the regions with colors corresponding to the class labels in** `Z`**.**
- **The** `alpha=0.75` **makes the contour plot semi-transparent.**
- `cmap=plt.cm.Paired` **sets the color map for the plot, using different colors for different regions.**

6. Plot the test data points:
plt.scatter(X_test[:, 0], X_test[:, 1], c=Y_test, edgecolors='k', marker='o', cmap=plt.cm.Paired)

`plt.scatter()` **is used to plot the test data points (**`X_test[:, 0]` **and** `X_test[:, 1]` **are the x and y coordinates of the test data).**
`c=Y_test` **colors the points based on their true labels from** `Y_test`**.**
`edgecolors='k'` **adds black edges around the points.**
`marker='o'` **sets the marker style to a circle.**

`cmap=plt.cm.Paired` **ensures that the same color map used in the decision boundary is applied to the data points.**