ort-vector-machines-assignment-1

September 13, 2023

```
[ ]: # Q1. What is the mathematical formula for a linear SVM?
```

The mathematical formula for a linear Support Vector Machine (SVM) can be represented as follows:

Given a dataset with N data points, each represented as a feature vector x of dimensionality D, and corresponding binary class labels y (where y is either +1 or -1), a linear SVM seeks to find a hyperplane represented by the equation:

$w \cdot x + b = 0$

Here:

“w” is the weight vector, which is perpendicular to the hyperplane and determines its orientation. “x” is the input feature vector. “b” is the bias term (also known as the intercept), which shifts the hyperplane away from the origin.

The goal of a linear SVM is to find the optimal “w” and “b” that maximize the margin between the two classes while minimizing classification errors. The margin is defined as the distance between the hyperplane and the nearest data points (support vectors) from each class.

The optimization problem for a linear SVM can be formulated as:

Minimize: $1/2\ \|w\|^2$

Solving this optimization problem leads to the determination of the optimal “w” and “b” values, which define the separating hyperplane for the linear SVM. Once these parameters are found, classification of new data points can be performed by evaluating the sign of $(w \cdot x + b)$, where a positive value corresponds to one class, and a negative value corresponds to the other class.

```
[ ]: # Q2. What is the objective function of a linear SVM?
```

The objective function of a linear Support Vector Machine (SVM) is formulated to find the parameters (weight vector “w” and bias term “b”) that define a hyperplane with the maximum margin between two classes while minimizing classification errors.

The primary objective function for a linear SVM is a convex optimization problem. The objective function can be expressed as follows:

Minimize: $1/2\ \|w\|^2$

Subject to: $y\ (w \cdot x\ + b)\quad 1$ for all data points

```
[ ]: # Q3. What is the kernel trick in SVM?
```

The kernel trick in SVM is a technique that allows SVMs to work with non-linear data by transforming it into a higher-dimensional space without explicitly calculating the transformation.

Instead, it uses a kernel function to measure similarity in the original space, enabling SVMs to find non-linear decision boundaries. Common kernels include the linear, polynomial, Gaussian (RBF), and sigmoid kernels, each suitable for different types of data.

```
[ ]: # Q4. What is the role of support vectors in SVM Explain with example ?
```

Support vectors are the data points closest to the decision boundary in a Support Vector Machine (SVM). They are essential because they define the position and orientation of the decision boundary, ensuring it maximizes the margin between classes while correctly classifying data. These support vectors are critical for the SVM's accuracy and robustness.

Here's an explanation with an example:

Imagine you have a dataset with two classes, represented by red and blue points on a 2D plane. You want to find a linear SVM classifier that can separate these two classes. The decision boundary (hyperplane) will be the line that best separates the data into two classes.

1 Step 1: Initially, the SVM algorithm tries to find the widest possible margin, which is the distance between the decision boundary and the nearest data points (support vectors) from each class.

2 Step 2: Support vectors are identified as the data points that are closest to the decision boundary. In the 2D plane, these support vectors will be the points from each class that are nearest to the decision boundary line. They are essentially the data points that define the margin.

3 Step 3: The decision boundary is then determined based on these support vectors. It is positioned so that it maximizes the margin while ensuring that it correctly classifies all other data points.

4 Step 4: Any changes in the position or orientation of the decision boundary depend on the support vectors. The other data points that are not support vectors do not influence the boundary's position, which makes the SVM robust to outliers or noisy data points.

In this example, the support vectors are critical because they define the SVM's decision boundary. Their positions ensure that the margin is maximized, and they help the SVM generalize well to new, unseen data points. By focusing on the support vectors, SVM achieves a form of regularization that prevents overfitting and contributes to its ability to handle complex datasets.

```
[ ]: # Q5. Illustrate with examples and graphs of Hyperplane, Marginal plane, Soft␣
     ↪margin and Hard margin in SVM.
```

In SVM, a hyperplane is a decision boundary that separates two classes. In a two-dimensional feature space (2D), it's a straight line; in a higher-dimensional space, it's a hyperplane.

Marginal Plane:

A marginal plane, also known as the margin boundary, is the region parallel to the hyperplane that defines the margin

Soft Margin:

In real-world datasets, perfect separation with a hard margin is often not possible due to noise or overlap between classes. The concept of a soft margin allows for some misclassification to find a better balance between maximizing the margin and minimizing errors.

Hard Margin:

A hard margin SVM aims to find a hyperplane that perfectly separates the two classes without any misclassification. It requires that all data points are correctly classified and lie outside the margin.

```
[2]: from IPython.display import Image

     # Specify The Image File Path
     # image_path = ''

     # Display the image
     # Image(filename=image_path)



     # Specify The URL Of The Image
     image_url = 'https://editor.analyticsvidhya.com/uploads/20470svm17.png'

     # Display The Image From The URL
     Image(url=image_url)
```

```
[2]: <IPython.core.display.Image object>
```

```
[3]: # Specify The URL Of The Image
     image_url = 'https://miro.medium.com/v2/resize:fit:2000/
     ↪1*l5yQxBwZJlwQN3jeGFG5ug.png'

     # Display The Image From The URL
     Image(url=image_url)
```

```
[3]: <IPython.core.display.Image object>
```

[ ]:
```
# Q6. SVM Implementation through Iris dataset.
# Load the iris dataset from the scikit-learn library and split it into a␣
↪training set and a testing setl
# Train a linear SVM classifier on the training set and predict the labels for␣
↪the testing setl
# Compute the accuracy of the model on the testing setl



# Plot the decision boundaries of the trained model using two of the featuresl
# Try different values of the regularisation parameter C and see how it affects␣
↪the performance of the model.


# Bonus task: Implement a linear SVM classifier from scratch using Python and␣
↪compare its performance with the scikit-learn implementation.
```

[20]:
```
# Import Necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import pandas as pd

# Load The Iris dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target


# Create A DataFrame To Display The Data
iris_df = pd.DataFrame(data=X, columns=iris.feature_names)
iris_df['Target'] = y




# Split The Dataset Into A Training Set And A Testing Set (70% Train, 30% Test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=40)

# Train A Linear SVM Classifier
svm_classifier = SVC(kernel='linear', C=1.0)  # You can adjust the C parameter
svm_classifier.fit(X_train, y_train)

# Predict The Labels For The Testing Set
```

```python
y_pred = svm_classifier.predict(X_test)

# Compute The Accuracy Of The Model On The Testing Set
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 1.00
```