Assignment 20 Solutions ¶

1. What is the underlying concept of Support Vector Machines ?

In [ ]:
```
The underlying concept of Support Vector Machines (SVM) is to find an optimal
classes by maximizing the margin or distance between the classes.
```

2. What is the concept of a support vector ?

In [ ]:
```
In Support Vector Machines (SVM), support vectors are the data points that lie
They play a crucial role in defining the decision boundary and determining the
These support vectors support or influence the placement and orientation of th
```

3. When using SVMs, why is it necessary to scale the inputs ?

```
Scaling the inputs is necessary when using Support Vector Machines (SVM) for
several reasons:

SVMs are sensitive to the scale of the input features. Features with larger
scales can dominate the optimization process and have a
disproportionate impact on the decision boundary.

Scaling ensures that all features contribute equally to the SVM model. By
bringing all features to a similar scale, the SVM can give
equal weight and importance to each feature during the training process.

Scaling helps improve the convergence of the optimization algorithm. It can
lead to faster convergence and more stable results.

Overall, scaling the inputs in SVMs helps prevent biased or skewed results
and ensures that the model can effectively learn from all
features in a fair and balanced manner.
```

4. When an SVM classifier classifies a case, can it output a confidence score? What about a percentage chance ?

```
Yes, an SVM classifier can provide a confidence score or a percentage chance
associated with its classification decision,
depending on the specific implementation and configuration.

In some SVM implementations, the distance from a data point to the decision
boundary can be used as a confidence score. A larger
distance indicates higher confidence in the classification, as the data
point is further away from the decision boundary.
```

Additionally, some SVM variants, such as the probabilistic SVM, provide a probability estimate or percentage chance for each class label. These models use techniques like Platt scaling or sigmoid calibration to map the SVM's output scores or distances to probabilities.

It's important to note that not all SVM implementations directly output probabilities or confidence scores by default. Some may require additional calibration or post-processing techniques to obtain probability estimates.

5. Should you train a model on a training set with millions of instances and hundreds of features using the primal or dual form of the SVM problem ?

When training a model on a training set with millions of instances and hundreds of features, it is generally recommended to use the dual form of the SVM problem.

The dual form of SVM is more computationally efficient in scenarios where the number of features is large compared to the number of instances. It avoids the need to explicitly calculate the feature vectors in the high-dimensional space, which can be computationally expensive.

By using the dual form, the SVM training process can be optimized to efficiently handle high-dimensional datasets. It leverages kernel functions to implicitly map the input data into a higher-dimensional space without explicitly calculating the transformed feature vectors.

However, it is important to note that the choice between the primal and dual form of SVM can depend on various factors, such as the specific implementation, the available computational resources, and the characteristics of the dataset. It is advisable to consider these factors and conduct empirical evaluations to determine the most suitable approach for a given scenario.

6. Let's say you've used an RBF kernel to train an SVM classifier, but it appears to underfit the training collection. Is it better to raise or lower (gamma)? What about the letter C ?

If an SVM classifier trained with an RBF (Radial Basis Function) kernel is underfitting the training data, you can consider adjusting the hyperparameters gamma and C.

Gamma (γ): Gamma determines the influence of a single training example on the decision boundary. To address underfitting, you can try increasing gamma. Higher gamma values make the SVM model more sensitive to individual data points, potentially leading to a more complex and flexible decision boundary.

C: C is the regularization parameter that controls the trade-off between achieving a larger margin and minimizing training errors.
To address underfitting, you can try decreasing C. Lower values of C allow for a wider margin and a more relaxed decision boundary,

```
which can better accommodate misclassified or overlapping training
instances.

In summary, for an SVM classifier with an RBF kernel that underfits the
training data, you can try raising gamma to make the model
more sensitive to individual data points and/or lower C to allow for a wider
margin and more flexibility in the decision boundary.
However, the exact values of gamma and C should be chosen through
experimentation and validation on a held-out dataset to find the
optimal balance and improve the model's performance.
```

7. To solve the soft margin linear SVM classifier problem with an off-the-shelf QP solver, how should the QP parameters (H, f, A, and b) be set ?

```
To solve the soft margin linear SVM classifier problem using an off-the-
shelf Quadratic Programming (QP) solver,
the QP parameters (H, f, A, and b) should be set as follows:

H: The Hessian matrix (or its approximation) represents the quadratic term
of the objective function. In the case of a soft margin SVM,
H is typically set as the identity matrix or a diagonal matrix with
regularization coefficients.

f: The f vector represents the linear term of the objective function. It
depends on the data and labels, and it is used to minimize
the classification errors. The values in the f vector depend on the
regularization parameters and the training data.

A: The A matrix represents the constraints on the decision boundary. For a
soft margin SVM, it includes both the inequality constraints
for the margin and the equality constraints for the misclassified instances.
The exact values and structure of the A matrix depend on
the training data, labels, and regularization parameters.

b: The b vector represents the right-hand side of the constraints. It
contains the margin and misclassification constraints' thresholds.
The values in the b vector depend on the training data, labels, and
regularization parameters.

It's worth noting that the exact formulation of the QP parameters may vary
slightly depending on the specific QP solver being used.
It is recommended to consult the documentation or guidelines provided by the
QP solver to ensure the correct parameter setting for
solving the soft margin linear SVM classifier problem.
```

8. On a linearly separable dataset, train a LinearSVC. Then, using the same dataset, train an SVC and an SGDClassifier. See if you can get them to make a model that is similar to yours ?

```
LinearSVC:

Import LinearSVC from the scikit-learn library.
Create an instance of LinearSVC and fit it to your linearly separable
dataset.
```

```
Retrieve the learned model parameters, such as coefficients and intercept.

SVC:

Import SVC from the scikit-learn library.
Create an instance of SVC with a linear kernel (e.g., kernel='linear') and
fit it to your linearly separable dataset.
Retrieve the learned model parameters, such as coefficients and intercept.

SGDClassifier:

Import SGDClassifier from the scikit-learn library.
Create an instance of SGDClassifier with a loss function suitable for linear
classification (e.g., loss='hinge') and fit it to your
linearly separable dataset.
Retrieve the learned model parameters, such as coefficients and intercept.
After training the models, you can compare the learned parameters
(coefficients and intercepts) across the three models.
If the dataset is indeed linearly separable, you would expect the models to
have similar coefficients and intercepts.

It's important to note that due to the inherent randomness in the training
process, the exact values of the model parameters may
differ slightly between the models. However, the overall pattern and
separation should be similar.

By comparing the models, you can assess their performance and see if they
provide similar decision boundaries for the linearly
separable dataset.
```

9. On the MNIST dataset, train an SVM classifier. You'll need to use one-versus-the-rest to assign all 10 digits because SVM classifiers are binary classifiers. To accelerate up the process, you might want to tune the hyperparameters using small validation sets. What level of precision can you achieve ?

```
When training an SVM classifier on the MNIST dataset, using the one-versus-
the-rest (OvR) strategy to classify all 10 digits,
it is possible to achieve a high level of precision. By tuning the
hyperparameters and utilizing small validation sets for faster
experimentation, you can optimize the performance of the SVM classifier.

The level of precision that can be achieved depends on various factors,
including the choice of SVM variant (linear, kernel-based),
hyperparameter settings (such as regularization parameter C and kernel
parameters), feature engineering techniques, and
the size and quality of the training dataset.

With proper hyperparameter tuning and feature engineering, it is possible to
achieve precision scores above 95% or even higher on
the MNIST dataset. However, the exact level of precision achieved will vary
based on the specific implementation and the strategies
employed during training and evaluation.
```

10. On the California housing dataset, train an SVM regressor ?

In [1]:

```
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error

# Load the California housing dataset
data = fetch_california_housing()

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data.data, data.target, te

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

# Create an SVM regressor
svr = SVR(kernel='rbf', C=1.0, epsilon=0.1)

# Train the SVM regressor
svr.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred = svr.predict(X_test_scaled)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

```
Mean Squared Error: 0.35700264267544685
```

In [ ]: