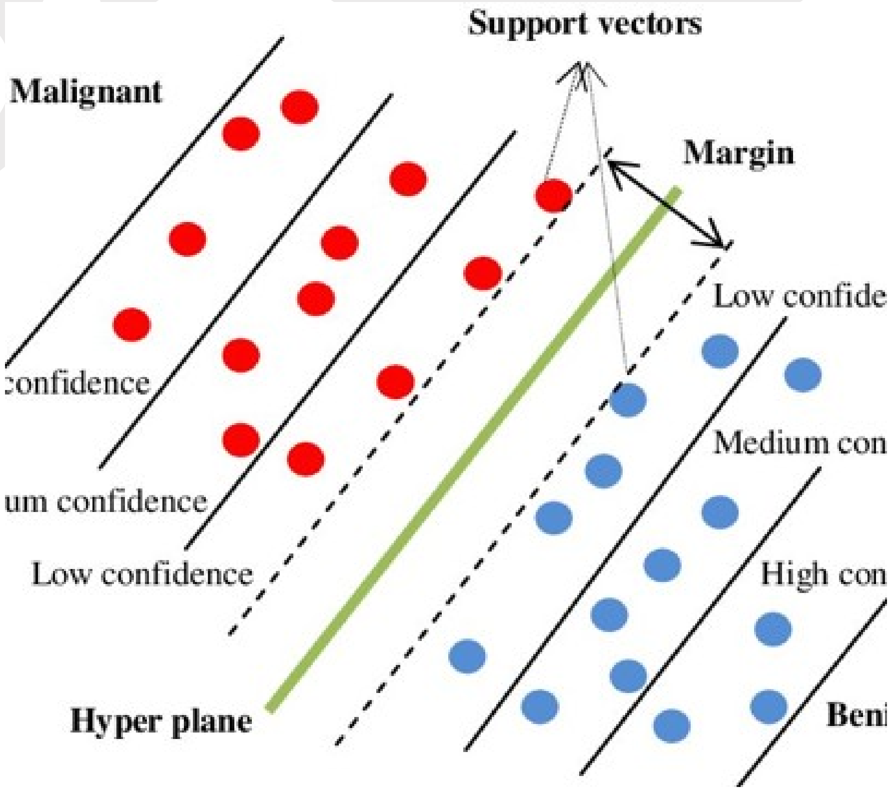
Coursework II – Handwritten Digits

February 2024

Classification using Support Vector Machine



- SVM starts by plotting all the data points on a hyperplane.
- Support Vectors are those data points that lie closest to the line separating each class cluster.
- The SVM classifier has to classify these points to their particular class with maximum accuracy.
- The distance between the Support Vectors and the decision boundary is called the Margin.
- The major task of any SVM classifier is to divide the data points is such manner that the Margin is maximum in value.

Multi Class Classification

Algorithms such as the ***Perceptron, Logistic Regression, and Support Vector Machines*** were designed for binary classification and do not natively support classification tasks with more than two classes.

How to use binary classification algorithms for multi-classification problems?

Split the multi-class classification dataset into multiple binary classification datasets and fit a binary classification model on each.

Two different examples of this approach are the **One-vs-Rest** and **One-vs-One** strategies.

- The One-vs-Rest strategy splits a multi-class classification into one binary classification problem per class.
- The One-vs-One strategy splits a multi-class classification into one binary classification problem per each pair of classes.

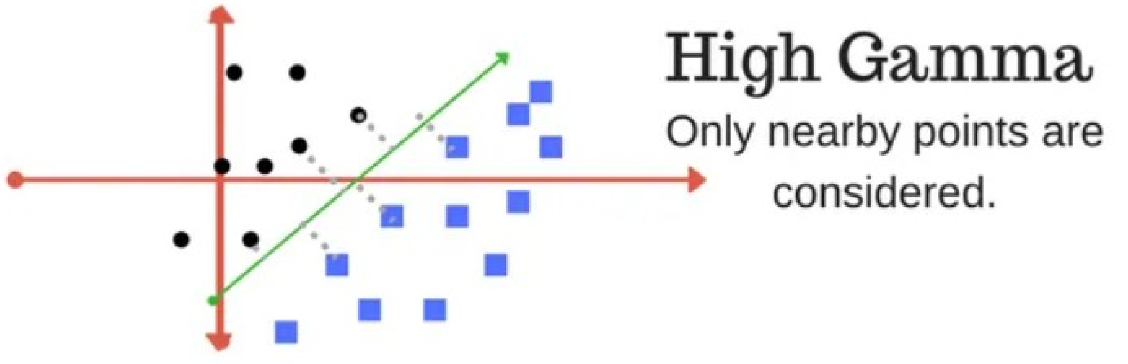
https://projector.tensorflow.org/

Tuning Parameters - **Kernel**

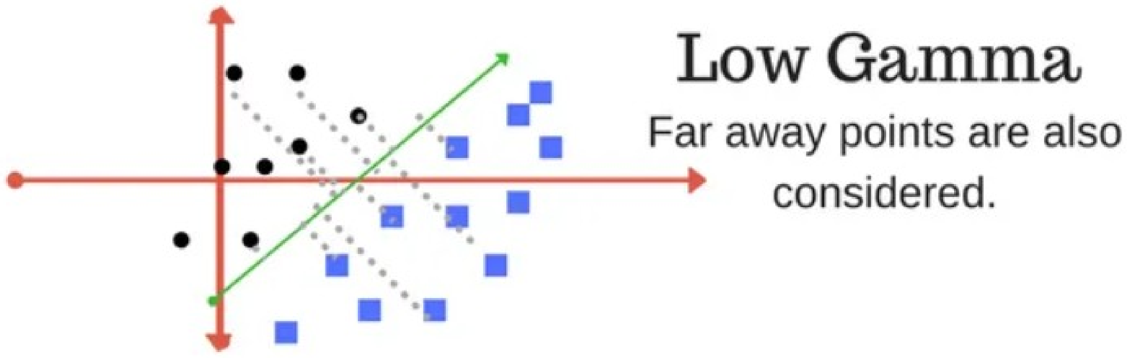
- Radial Basis Function or Gaussian Kernel
- A kernel is a function that returns the result of a dot product performed in another space.
  - **A kernel is a measure of the similarity between two vectors**
- A RBF is a function whose value depends only on the distance from the origin or from some point.
  - The RBF kernel returns the result of a dot product performed in $\mathbb{R}^{\infty}$.

$$K(\mathbf{x}, \mathbf{x}') = \exp(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2)$$

Tuning Parameters - Gamma



High Gamma

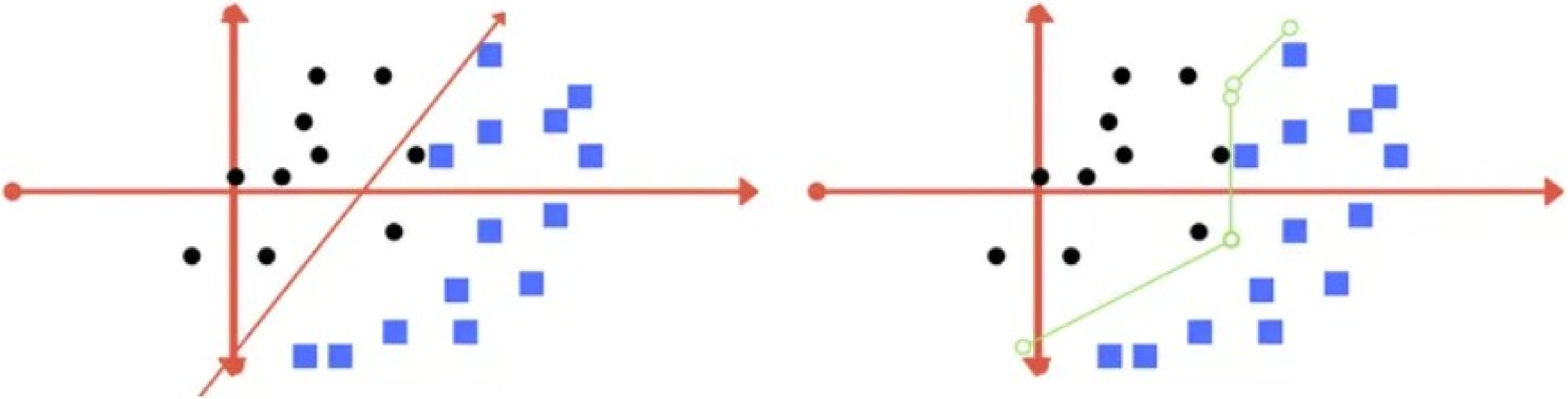


Low Gamma

- The gamma parameter defines how far the influence of a single training example reaches, with low values meaning ‘far’ and high values meaning ‘close’.
- With low gamma, points far away from plausible separation line are considered in calculation for the separation line.
- High gamma means the points close to plausible line are considered in calculation.
- Gamma = 0.5

Tuning Parameters - **Regularization**

- The Regularization parameter (often termed as C parameter in python's sklearn library) tells the SVM optimization how much you want to avoid misclassifying each training example.
- For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly.
- Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points.
- C = 0.01



Left: low regularization value, right: high regularization value

Main Function

- Read Train Dataset
- Read Test Dataset
- Build 10 SVMs
  - Train for 10 classes
- Calculate accuracies for train and test

Constructor for SVM

```
/**
* @param name_ name of SVM
* @param data_ training dataset
* @param dataClass_ expected values of training dataset
* @param C_ regularization hyperparameter
* @param gamma_ hyperparameter for RBF
* @param positiveData_ integer that will be considered positive output
*/
```

trainModel()

1. **Initialization:**

   - Initialize variables:

     - `numChanged` to 0, which keeps track of the number of alphas changed during the iteration.

     - `examineAll` to `true`, indicating that the algorithm will start by examining all data points.

     - `iterations` to 0, representing the current iteration count.

   - Enter a loop that continues until either the maximum number of iterations (`MAX_ITERATIONS`) is reached or no alphas are changed, and `examineAll` is `false`.

2. **Iteration:**

   - Print the current iteration number for monitoring purposes.

   - Set `numChanged` to 0 at the beginning of each iteration.

3. **Optimization Loop:**

   - Depending on the value of `examineAll`, iterate through all data points (`numSamples`) or only those with alphas within the (0, C) range.

   - For each data point, call the `optimizeAlpha` method to attempt to optimize the Lagrange multiplier (`alpha`). The method returns 1 if an optimization occurs and 0 otherwise.

   - Update `numChanged` by adding the returned value from `optimizeAlpha`.

4. **Update `examineAll`:**

   - Update the value of `examineAll` based on the results of the iteration:

     - If `examineAll` was `true` and `numChanged` is greater than 0, set `examineAll` to `false`.

     - If `examineAll` was `false` and `numChanged` is 0, set `examineAll` to `true`.

5. **Increment Iteration Count:**

   - Increment the `iterations` counter.

6. **Loop Termination:**

   - Continue the loop until either the maximum number of iterations is reached or no alphas are changed, and `examineAll` is `false`.

Optimizing the values of the Lagrange multipliers (alphas) for a pair of data points in order to update the SVM model.

1. **Initialization:**

- `y1` is the label of the first data point.

- `alpha1` is the Lagrange multiplier (weight) associated with the first data point.

- `E1` is the error of the SVM prediction for the first data point.

2. **Check Conditions:**

- Checks whether the current Lagrange multiplier `alpha1` violates the KKT (Karush-Kuhn-Tucker) conditions, which are necessary for the solution of the SVM optimization problem.

- If conditions are not met, the method returns 0, indicating that no optimization is performed.

3. **Choose a Second Data Point:**

- `i2` is chosen randomly, and its label (`y2`), Lagrange multiplier (`alpha2`), and error (`E2`) are obtained.

4. **Compute Bounds (`L` and `H`):**

- Computes the lower bound (`L`) and upper bound (`H`) for the new Lagrange multiplier (`alpha2`) based on the SVM optimization problem constraints.

5. **Compute Kernel Values:**

- Computes the kernel values `k11`, `k22`, and `k12` using the radial basis function (RBF) kernel for the chosen data points.

6. **Update Lagrange Multipliers:**

- Updates the Lagrange multiplier `alpha2` based on the chosen optimization strategy.

- Handles cases where the new value of `alpha2` is outside the computed bounds.

7. **Check Convergence:**

- Checks for convergence based on a tolerance condition. If the change in `alpha2` is small, no further optimization is performed.

8. **Update Bias (`b`):**

- Computes the bias term (`b`) based on the updated Lagrange multipliers.

- If either `a1` or `a2` is within the (0, C) range (C is a regularization parameter), `b` is set to the corresponding value (`b1` or `b2`). Otherwise, it is set to the average.

9. **Update Lagrange Multipliers in the Array (`alphas`):**

- Updates the Lagrange multipliers for the chosen data points in the array `alphas`.

10. **Return Status:**

- Returns 1 to indicate that the optimization was successful.

11. **Fallback:**

- If the initial conditions are not met, the method returns 0, indicating that no optimization was performed.

Other useful helper methods

- Choose a random index
- rbfKernel implementer – calculates the dot product of x1 and x2 in infinite dimensions
- Calculate error in the value of alpha
- Classify to calculate whether a datapoint lies with respect to a hyperplane
- Calculate the value of the quadratic problem to optimise for alpha