Machine Learning- Exercise 3

AdaBoost

Stephanie Kaes & George Lydakis
<lastname>@vision.rwth-aachen.de

RWTH Aachen University - Computer Vision Group
http://www.vision.rwth-aachen.de/

2023-12-06

Content

1. Cross-Validation
2. AdaBoost Recap
3. Exercise 3: AdaBoost
   a) Simple Classifier
   b) Simple AdaBoost
   c) Cross-Validation AdaBoost
   d) Least Squares AdaBoost
   e) Application

Cross Validation

Visual Computing Institute | RWTH AACHEN UNIVERSITY

Two variants of CV are very popular:

Hold-out CV

Typically 80/20 i.e.

- Use 80% of the data for training
- Use 20% for testing

For ex.: *sklearn train-test-split function*

→No validation data

→Requires quite large datasets

→Problematic if data is imbalanced

K-folds CV

1. Split the training dataset into k equally sized training data and validation data subsets.
2. Train the model with the training data.
3. Validate the performance of the model using the validation data.
4. Repeat steps 1-3 or break

…but there are many more!

Cross Validation

How do we apply k-folds CV ?

1. Split the training dataset into k equally sized training data and validation data subsets.
2. Train the model with the training data.
3. Validate the performance of the model using the validation data.
4. Repeat steps 1-3 (5-10 times) or break

Cross Validation

Benefits of k-folds CV?

- **Reduces Overfitting:** If a model consistently performs well across various subsets, it is more likely to generalize well.
- **Maximizes Data Utilization:** Instead of using only one fixed split, we use multiple combinations, ensuring that each data point is used for both training and testing at some point. Can be useful if test dataset is really small or data shows class imbalances.
- **Enhances Robustness:** Averaging across multiple folds reduces the impact of randomness in a single train-test split -> more stable performance estimate.

Cross Validation

Further Benefits and Applications of k-folds CV:

- **Model Performance Estimation:** Better assessment of model‘s performance on unknown data than single-split
- **Hyperparameter Tuning:** Selecting the best set of hyperparameters by assessing the model's performance across different parameter configurations using random/grid search etc.
- **Model Selection:** If multiple models shall be compared: Select the best-performing model by comparing the average performance across folds

Cross Validation

Warning

- k-folds CV migh require many ressources
- Be careful when datasets show an inherent order (time-series data etc.) – k-folds CV might not be applicable here
- k-folds CV assumes that the data points are independent and equally distributed which might not be true

How to use CV

1. **Data Preparation**
    1. Split your dataset into features and target variable
    2. Apply further preprocessing if necessary
2. **Select a Cross-Validation Technique**
3. **Model Training and Evaluation Loop**
   For each fold or iteration of the cross-validation:
    - Split the data into training and testing sets.
    - Train your model on the training set.
    - Evaluate the model on the testing set (apply your desired performance metrics)
4. **Aggregate Results**
   Collect the performance metrics obtained from each fold or iteration.
5. **Compute Average Performance**
   Calculate the average of the performance metrics obtained from all folds.
6. **Analyze Results**

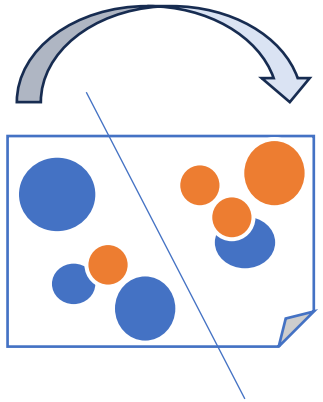
AdaBoost: Main Idea

Components

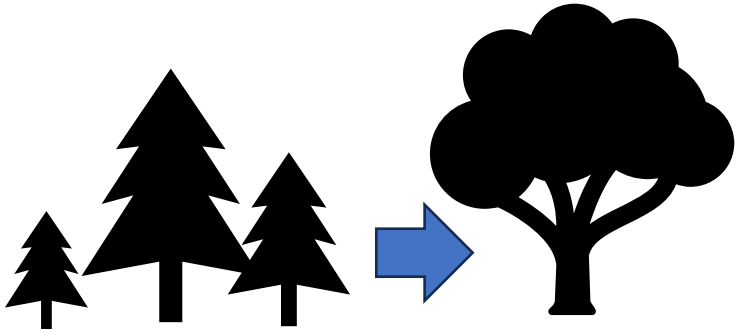
- $c_k(\mathbf{x})$: "weak" or base classifier
  - Condition: $< 50\%$ training error over any distribution
- $C(\mathbf{x})$:"strong or final classifier

Set of weak classifiers

at least better than random choice



Train & focus on misclassified data



Weighted sum

Final classifier

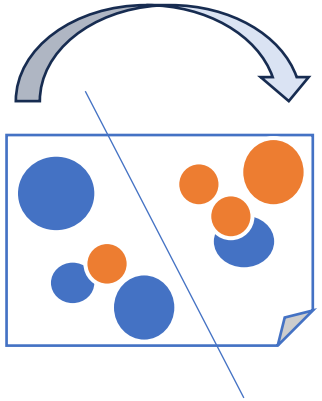
classifiers weighted with α_k

AdaBoost: Main Idea

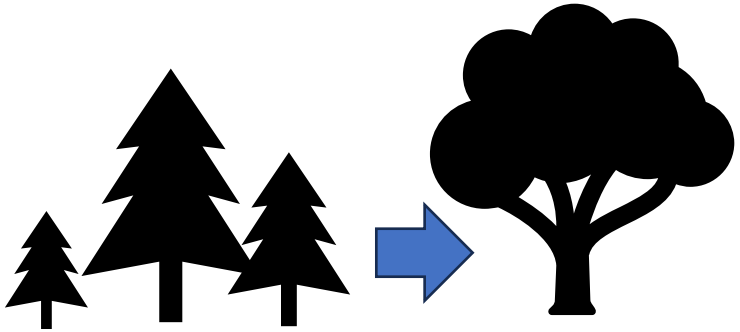
- Combine a weighted set of weak classifiers
- Resulting classifier should be better than the weak classifiers („Ensemble Learning“)

Set of weak classifiers

at least better than random choice



Train & focus on misclassified data



Weighted sum

Final classifier

classifiers weighted with α_k

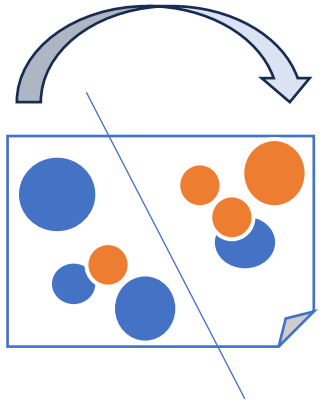
AdaBoost: Main Idea

Construct a strong classifier as a thresholded linear combination of the weighted classifiers:

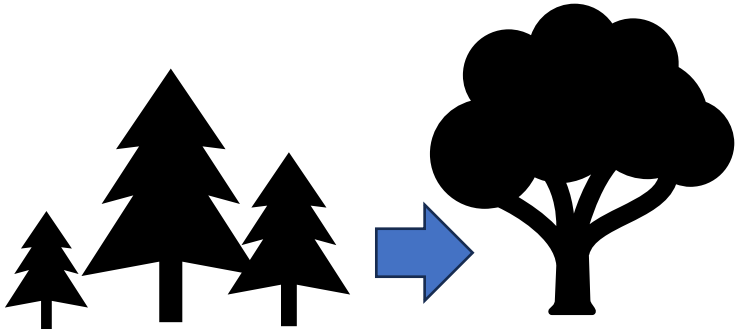
$$C(\mathbf{x}) = \text{sign}\left(\sum_{k=1}^{K} \alpha_k c_k(\mathbf{x})\right)$$

Set of weak classifiers

at least better than random choice



Train & focus on misclassified data



Weighted sum

Final classifier

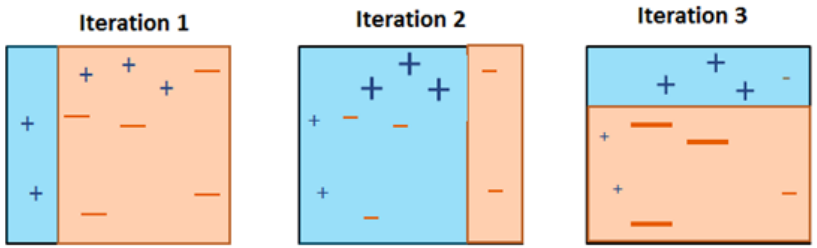
classifiers weighted with α_k

AdaBoost: Main Idea

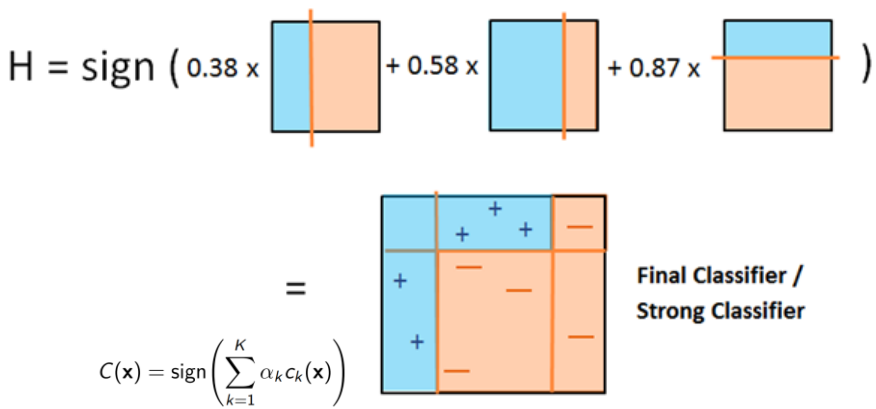
Visual Computing Institute | RWTH AACHEN UNIVERSITY

Main idea

- Instead of resampling, reweight misclassified training examples.
  - Increase the chance of being selected in a sampled training set.
  - Or increase the misclassification cost when training on the full set.



Data points weighted with $w_i^{(k)}$

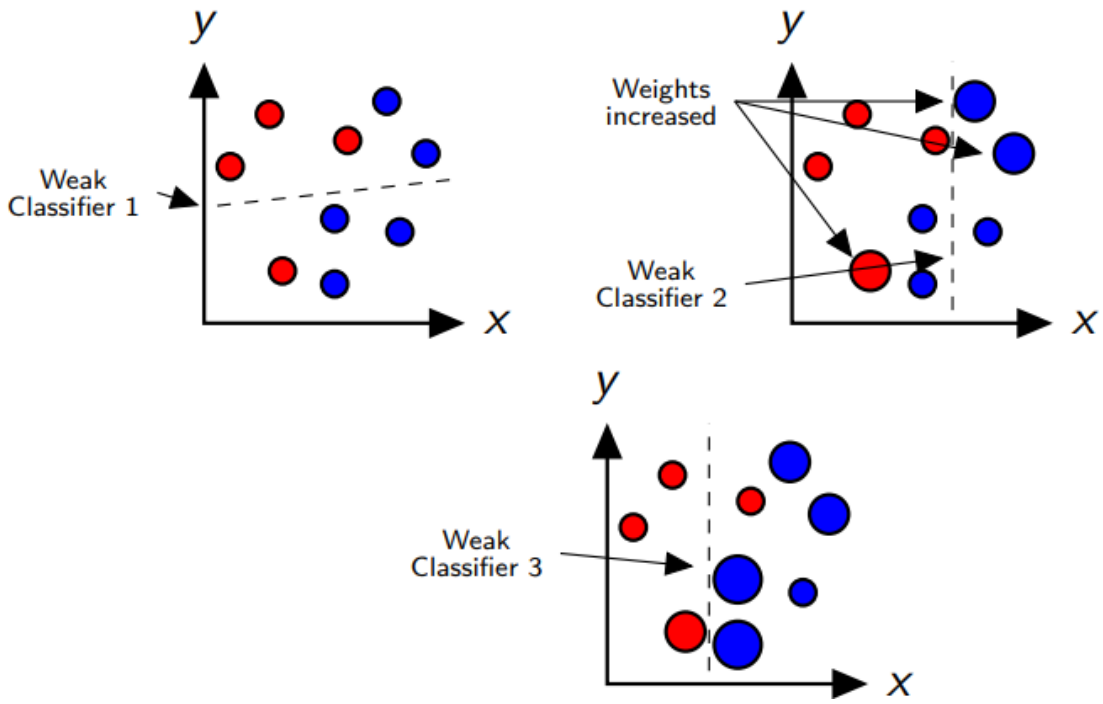


AdaBoost: Boosting Part

Consider a 2D feature space with positive and negative examples.

Each weak classifier splits the training examples with at least 50% accuracy.

Examples misclassified by a previous weak learner are given more emphasis at future rounds.



Final classifier is combination of the weak classifiers.

Image: Numerical Computing with Python (packtpub.com)

AdaBoost: Boosting Part

How do we emphasize wrongly classified data points?

If previously misclassified: Increase probability to be chosen as training data for upcoming weak classifier

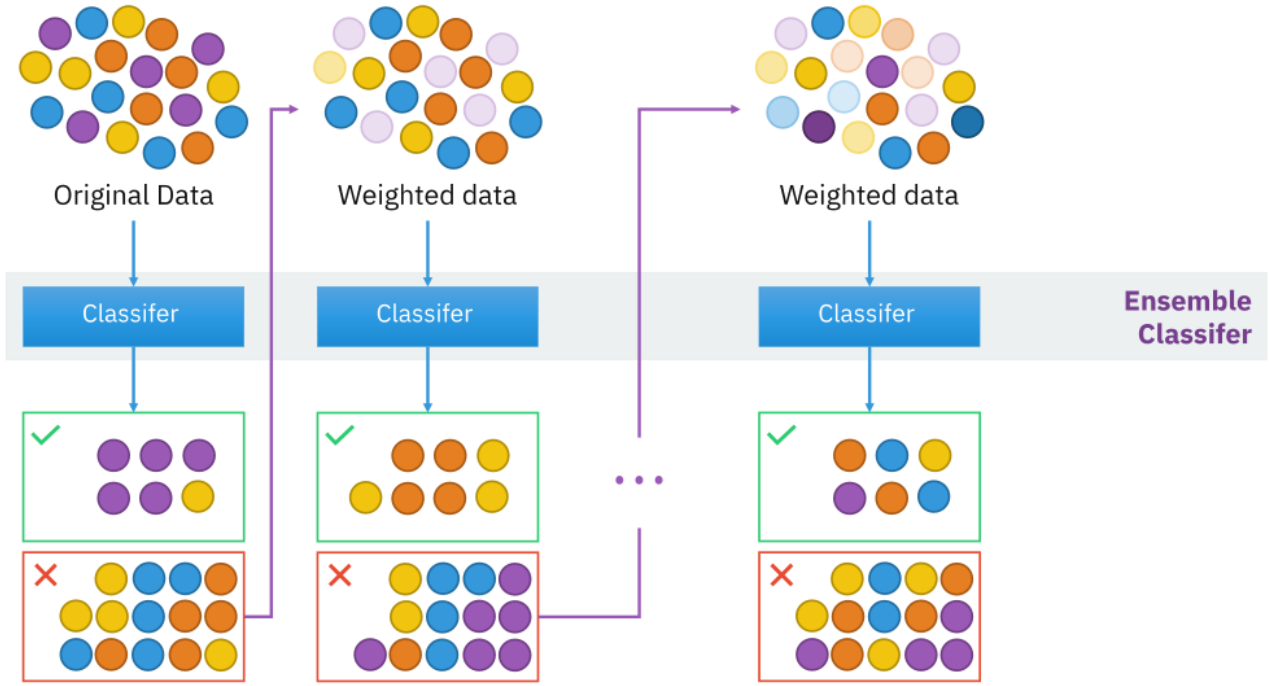


Image: A Comparitive Study Between AdaBoost and Gradient Boost ML Algorithm (analyticsvidhya.com)

AdaBoost: Algorithm

Visual Computing Institute | RWTH AACHEN UNIVERSITY

- Initialization: Set $w_n^{(1)} = \frac{1}{N}$ for $n = 1, \ldots, N$.
- For $k = 1, \ldots, k$ iterations
  - Train a new weak classifier $c_k(\mathbf{X})$ using current weights $\mathbf{W}^{(k)}$ by minimizing the weighted error function
  - estimate the weighted error of this classifier on $\mathbf{X}$:

$$\epsilon_k = \frac{\sum_{n=1}^{N} w_n^{(k)} I(c_k(\mathbf{X}) \neq y_n)}{\sum_{n=1}^{N} w_n^{(k)}}$$

  - Calculate a weighting coefficient for $c_k(\mathbf{X})$:

$$\alpha_k = \ln \left\{ \frac{1 - \epsilon_k}{\epsilon_k} \right\}$$

  - Update the weighting coefficients:

$$w_n^{(k+1)} = w_n^{(k)} \exp\{\alpha_k I(c_k(\mathbf{X}) \neq y_n)\}$$

Image: Numerical Computing with Python (packtpub.com)

Visual Computing Institute | RWTH AACHEN UNIVERSITY

Exercise 3

Solution

Solution - Exercise 3a

(a) When AdaBoost is applied to a weak learner with a linear decision boundary, what is the form of the resulting decision boundary of the boosted classifier?

All single classifiers create linear boundaries

→ Final classifier will be a piecewise linear function

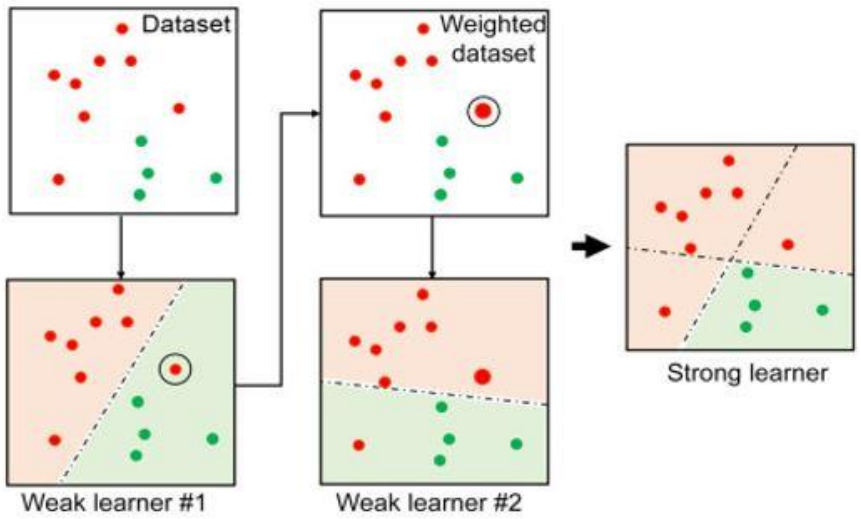


Image: Introduction to AdaBoost for Absolute Beginners - Analytics Vidhya

Solution - Exercise 3b

Code: Simple Classifier Function

RWTH AACHEN UNIVERSITY

Machine Learning WS 2023

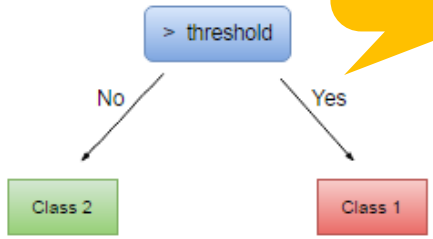
(b) As you know from the lecture, the foundation of the Adaboost algorithm are the so-called weak classifiers. For a start we are going to use extremely simple classifiers as weak classifiers.

threshold value

dimension

$$c(\mathbf{x}|j,\theta,p) := \begin{cases} 1 & \text{if } px_j > p\theta \\ -1 & \text{otherwise,} \end{cases}$$

Decision stump



where $p \in \{-1, 1\}$ is a parity indicating the direction of the inequality sign. This classifier ignores all entries of $\mathbf{x}$ except those in dimension j, x_j. To train this weak classifier on weighted data, we use the following learning rule:

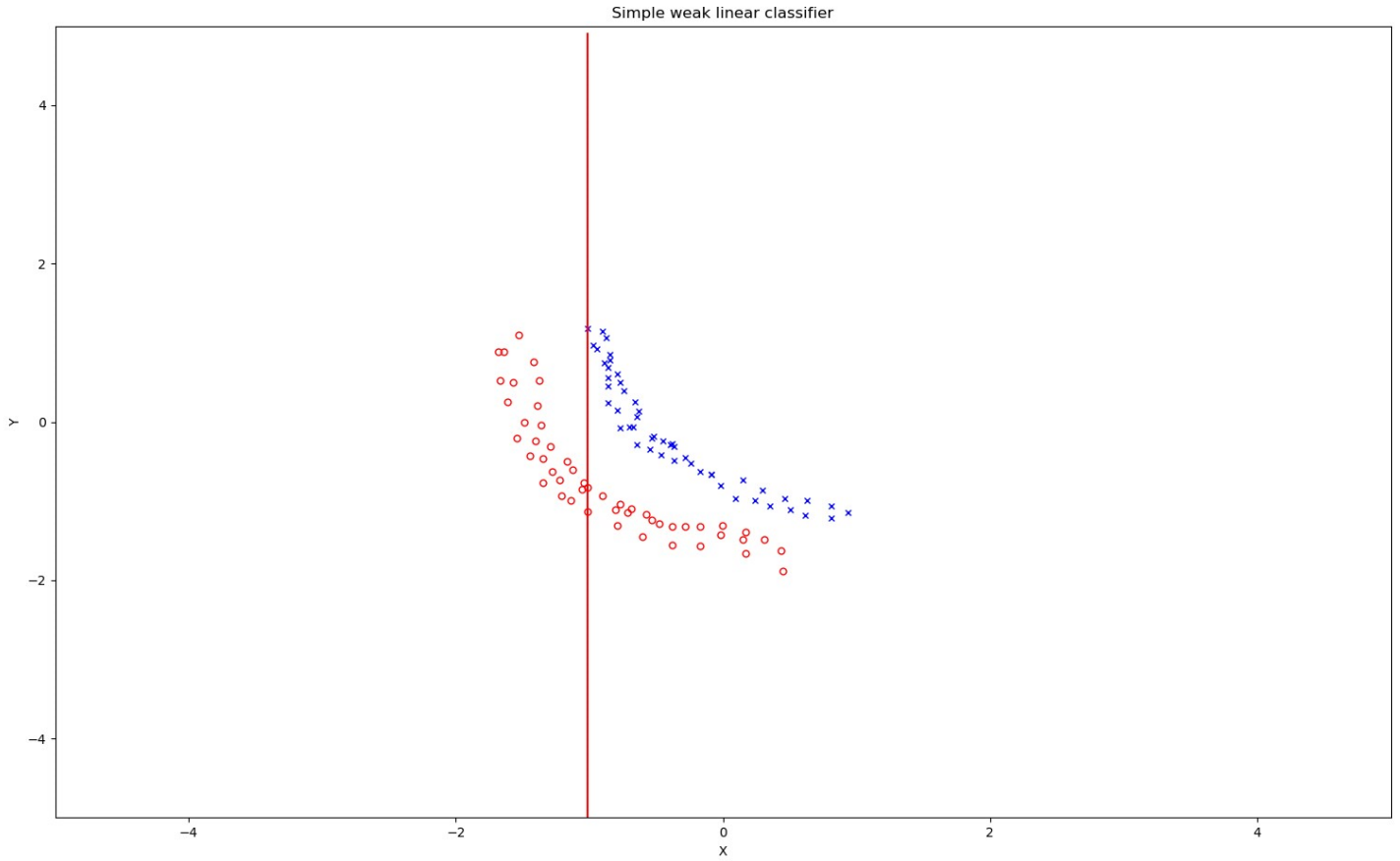
$$(\hat{j}, \hat{\theta}) := \text{argmin}_{j,\theta} \frac{\sum_{i=1}^{n} w_i \mathbf{I}\{y_i \neq c(\mathbf{x}|j,\theta,p)\}}{\sum_{i=1}^{n} w_i}$$

We count how often this is true.

For each dimension, there is an optimal decision boundary θ. We aim to minimize this function, i.e. we try to find a classifier where the number of misclassified data points is minimal.

Image: Machine Learning with Boosting - EOF (jasonleaster.github.io)

Solution - Exercise 3b



Solution - Exercise 3c

Code: AdaBoost Simple + Eval


```python
def simpleClassifier(X, Y):
    # Select a simple classifier
    #
    # INPUT:
    # X         : training examples (numSamples x numDim)
    # Y         : training lables (numSamples x 1)
    #
    # OUTPUT:
    # theta     : threshold value for the decision (scalar)
    # j         : the dimension to "look at" (scalar)

    #####Start Subtask 1b#####
    N, D = X.shape

    # Initialize least error
    le = 1
    j = 1  # dimension
    theta = 0  # decision value

    # Iterate over dimensions, which j to choose
    for jj in range(D):

        # Find interval to choose theta
        val = X[:, jj]  # shape: (100 x 1)

        sVal = np.sort(val) # TODO: returns unique sorted values, shape: (100 x 1)
        idx = np.argsort(val)
        change = np.where(np.roll(Y[idx], -1) + Y[idx] == 0)[0]  # shape: (36 x 1)

        # Calculate thresholds for which we want to check classifier error.
        # Candidates for theta are always between two points of different classes.

        th = (sVal[change[change < len(X)-1]] + sVal[change[change < len(X)-1]+1])/2  # shape: (35 x 1)

        error = np.zeros(len(th))  # error-placeholder for each value of threshold th
```

where class labels change, we have potential Θ values

potential Θ values

Solution - Exercise 3c

Code: AdaBoost Simple + Eval

RWTH AACHEN UNIVERSITY

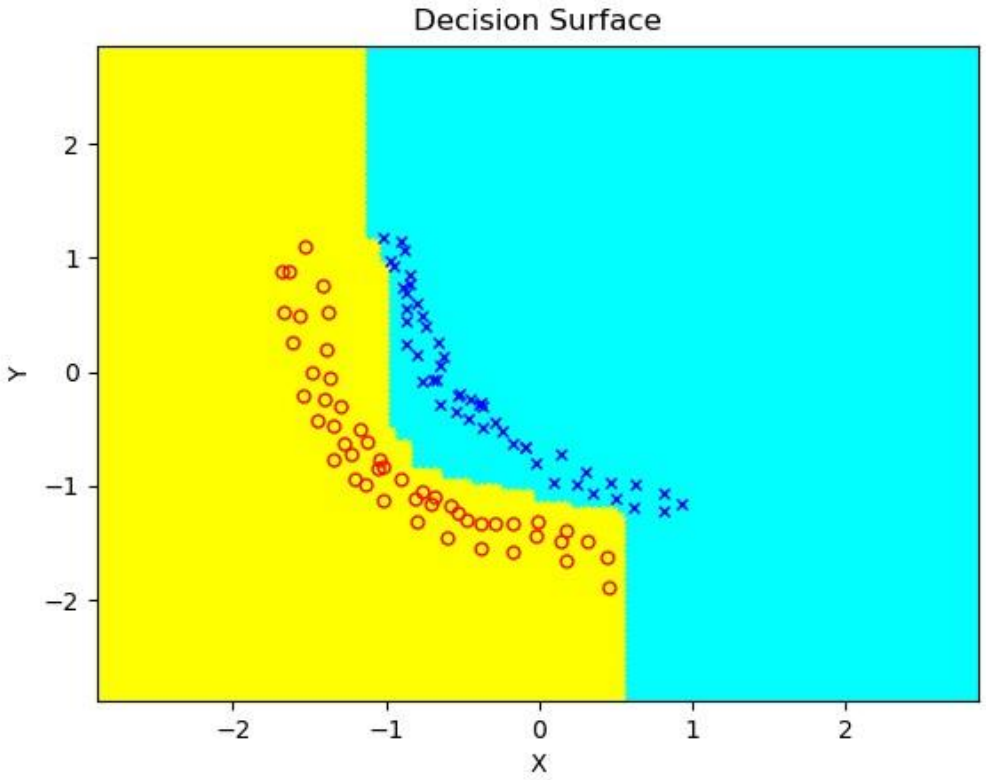
```
        # Iterate over canidates for theta
        for t in range(len(th)):
            # Initialize temporary labels for given j and theta
            cY = np.ones(N)*(-1)  # shape: (100 x 1)

            # Classify
            cY[X[:, jj] > th[t]] = 1  # Set all values to one, which are bigger then current threshold

            # Calculate error for given classifier
            error[t] = sum([Y[i] != cY[i] for i in range(N)])

            # Visualize potential threshold values
            print('J = {0} \t Theta = {1} \t Error = {2}\n'.format(jj, th[t], error[t]))

        le1 = min(error/N)
        ind1 = np.argmin(error/N)
        le2 = min(1-error/N)
        ind2 = np.argmin(1-error/N)
        le0 = min([le1,le2,le])

        if le == le0:
            continue
        else:
            le = le0
            j=jj+1  # Set theta to current value of threshold
            # Choose theta and parity for minimum error
            if le1 == le:
                theta = th[ind1]
            else:
                theta = th[ind2]

    #####End Subtask#####
    return j, theta
```

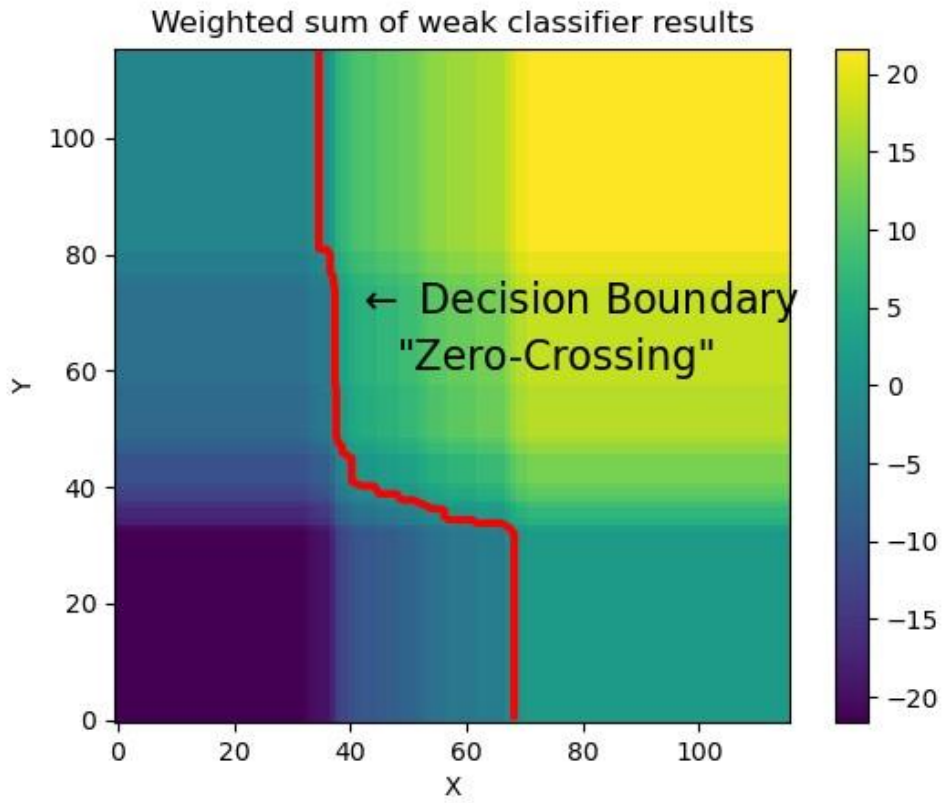
Solution - Exercise 3c

A quick recap of the AdaBoost algorithm: Given a weak classifier and training samples (x_i, y_i), $i = 1 \dots N$ (where $x_i \in \mathbb{R}^d$ are data points in d dimensional space and $y_i \in \{-1, +1\}$ are class labels), the AdaBoost algorithm for the two-class problem is:

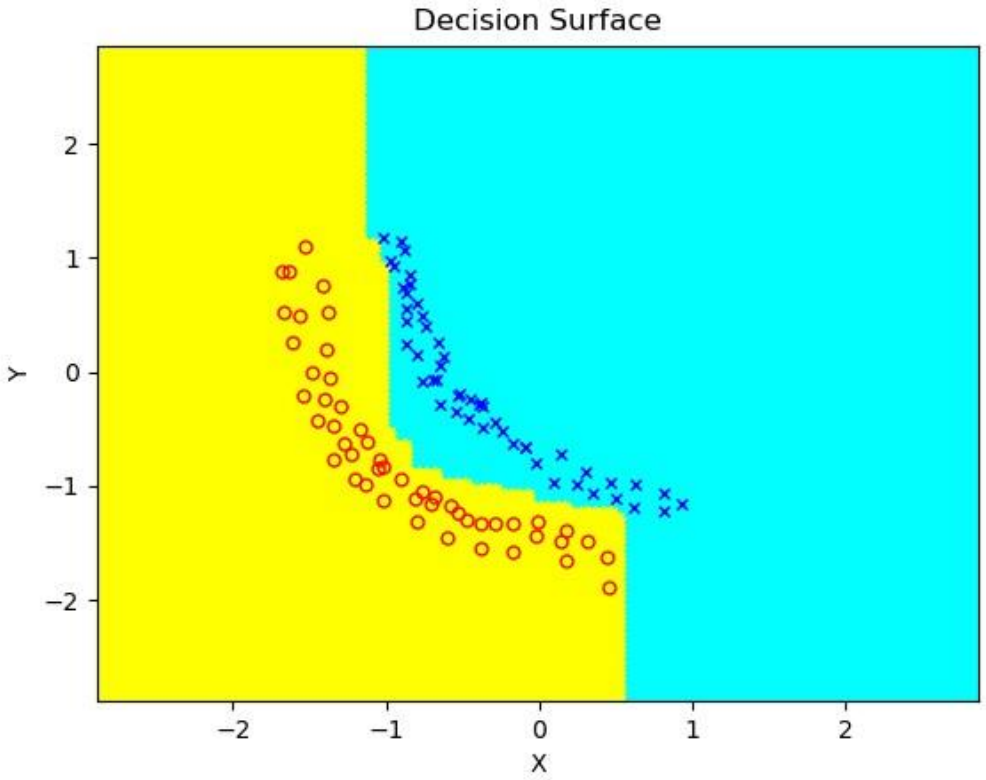
1. Initialize the weights $w_i^1 = \frac{1}{N}$
2. For $k = 1 \dots K$
   (a) Train the weak classifier c_k on the weighted data i.e. select the optimal parameter combination
   (b) Compute the error: $e_k = \frac{\sum_{i=1}^{N} w_i^k \mathbf{I}\{y_i \neq c_k(x_i)\}}{\sum_{i=1}^{N} w_i^k}$
   (c) Compute the voting weight for the weak classifier: $\alpha_k = \frac{1}{2} ln\left(\frac{1-e_k}{e_k}\right)$
   (d) Recalculate the weights: $w_i^{k+1} = w_i^k \exp\left(-\alpha_k y_i c_k(x_i)\right)$ and normalize w such that it is a probability distribution
3. Resulting classifier: $c_K(\mathbf{x}) = sgn\left(\sum_{k=1}^{K} \alpha_k c_k(\mathbf{x})\right)$;

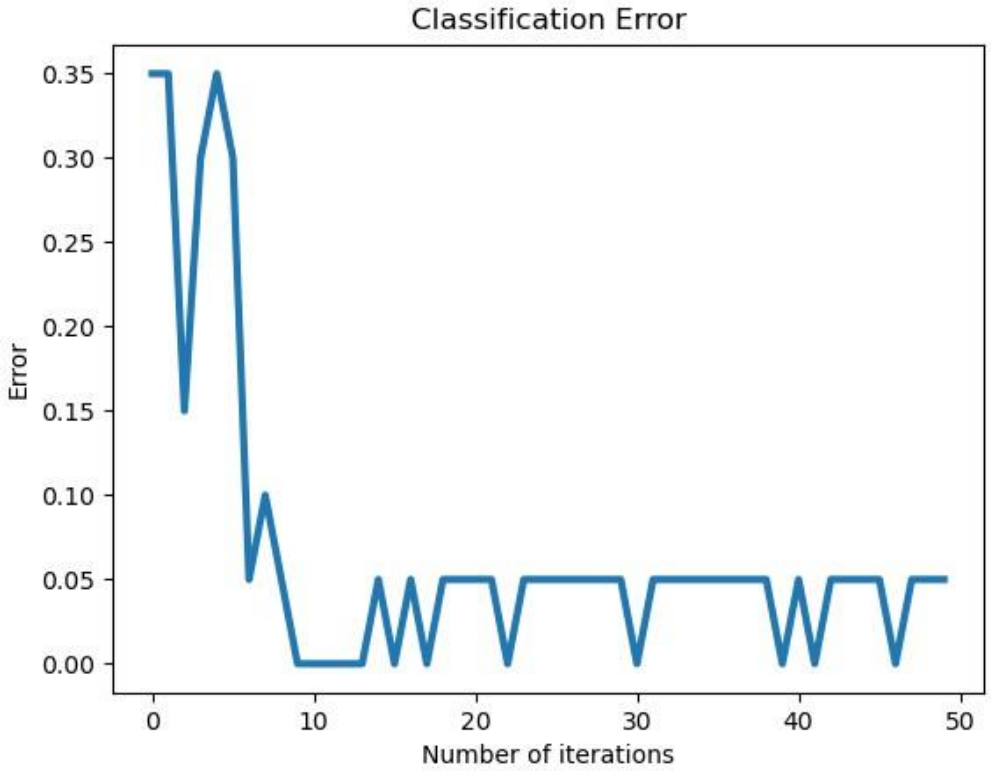
Solution - Exercise 3c





Solution - Exercise 3c

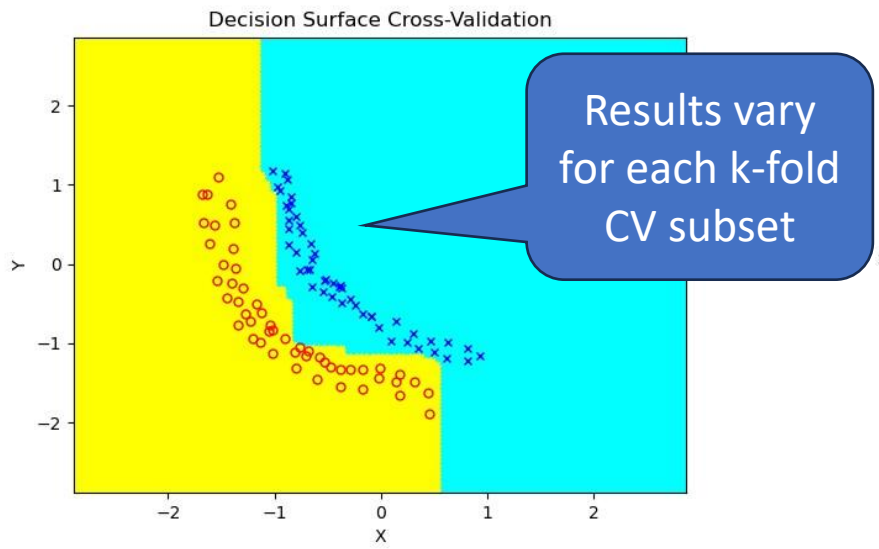


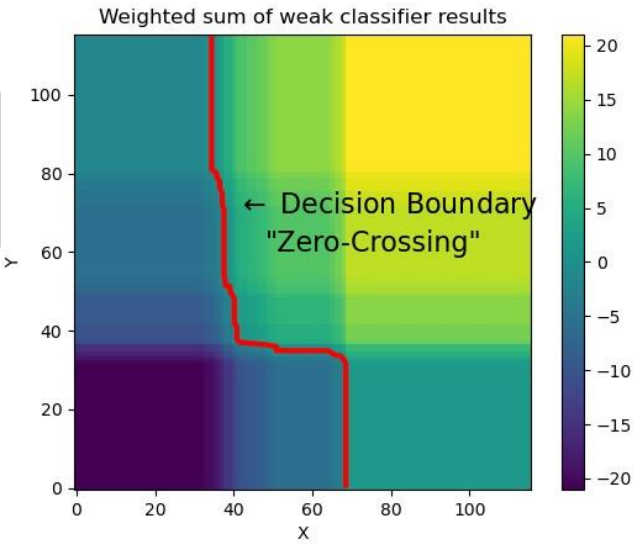


Solution - Exercise 3d

Code: AdaBoost Cross

(d) Next, we are going to add a cross-validation step to the training procedure. This means that only a part of the available training data is used for actual training. The remaining part is used to estimate the generalization performance of the learned classifier. To this end, split the training data according to the given percentage (percent = 0.4 means 40% is used for **validation**). After each iteration k of the algorithm, estimate the classification error of the current boosting classifier (not the base classifier) by cross-validation.





Solution - Exercise 3d

Plot the cross validation error estimates vs. the number k of iteration. What do you observe?

You can see that the decision boundaries vary due to the differences in the training datasets

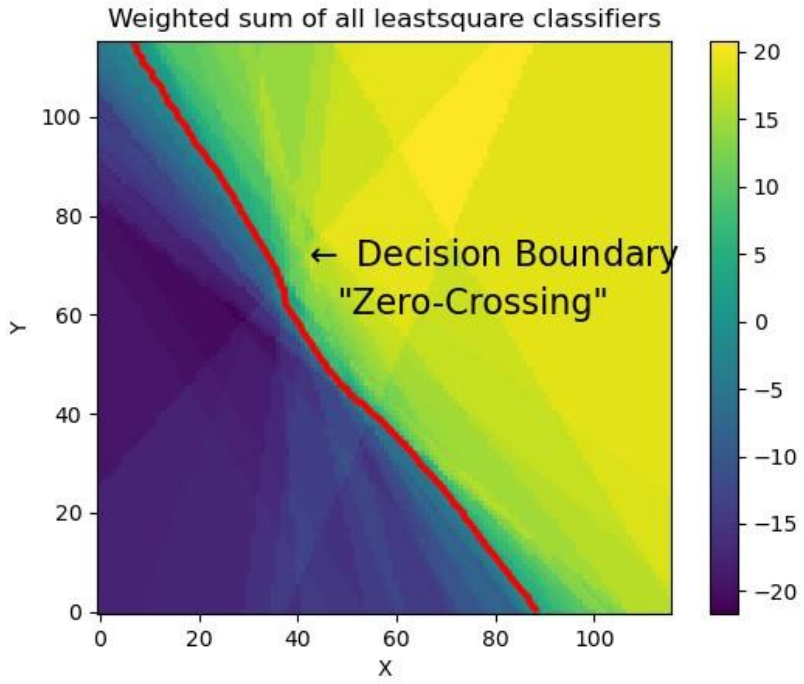
Why do we apply Cross Validation?

- Prevent overfitting
- Find optimal parameter set

Solution - Exercise 3e

Code: AdaBoost LSLC + Eval

(e) Now, we are going to use a more complex weak classifier in the AdaBoost framework, namely the least squares classifier (already implemented in `leastSquare`). Implement the AdaBoost algorithm in



Solution - Exercise 3f

Code: AdaBoost USPS

(f) Now, you are going to use the least-squares based AdaBoost on real data, i.e. the USPS data (provided in `usps.mat`). The dataset consists of a matrix `X` and a label vector `Y`. Each row of the matrix `X` is an image of size 20×14 and can be viewed with `matplotlib.pyplot.imshow(X[0,:].reshape(20,14,order='F').copy())`. The first 5000 rows of X contain the images of the digit 2, and the rest contains the images of the digit 9. Perform a random split of the 10000 data points into two equally sized subsets, one for training and one for validation. Run this at least three times and plot the cross validation error estimates vs. the number k of iterations.

"2" vs. "9"