IML (67577) - Exercise 4 - Boosting and SVM

Alon Emanuel - 205894058

May 24, 2019

SVM - Formulation

Q1

- We claim that the following QP problem's objective is equivalent to the Hard-SVM objective:

$$\underset{\mathbf{v}\in\mathbb{R}^n}{\operatorname{argmin}} \begin{bmatrix} \frac{1}{2}\begin{bmatrix} | \\ \mathbf{w} \\ | \\ b \end{bmatrix}^T \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \\ & & & & 0 \end{bmatrix} \begin{bmatrix} | \\ \mathbf{w} \\ | \\ b \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}^T \begin{bmatrix} | \\ \mathbf{w} \\ | \\ b \end{bmatrix} \end{bmatrix}$$

$$\text{s.t} \begin{bmatrix} - & -\mathbf{x}_1 & - & -1 \\ - & -\mathbf{x}_2 & - & -1 \\ & \vdots & & \\ - & -\mathbf{x}_m & - & -1 \end{bmatrix} \begin{bmatrix} | \\ \mathbf{w} \\ | \\ b \end{bmatrix} \leq \begin{bmatrix} -1 \\ -1 \\ \vdots \\ -1 \end{bmatrix}$$

- $Q = \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ & & & & 0 \end{bmatrix}$, $\mathbf{v} = \begin{bmatrix} | \\ \mathbf{w} \\ | \\ b \end{bmatrix}$, $\mathbf{a} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$, $A = \begin{bmatrix} - & -\mathbf{x}_1 & - & -1 \\ - & -\mathbf{x}_2 & - & -1 \\ & \vdots & & \\ - & -\mathbf{x}_m & - & -1 \end{bmatrix}$, $\mathbf{d} = \begin{bmatrix} -1 \\ -1 \\ \vdots \\ -1 \end{bmatrix}$.

- **Proof:**
  - Let $\mathbf{w}^*$ and b^* be some optimal solutions for the original Hard-SVM problem.
  - Lets plug it into our new QP objective:

$$\frac{1}{2}\begin{bmatrix} | \\ \mathbf{w}^* \\ | \\ b^* \end{bmatrix}^T \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \\ & & & & 0 \end{bmatrix} \begin{bmatrix} | \\ \mathbf{w}^* \\ | \\ b^* \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}^T \begin{bmatrix} | \\ \mathbf{w}^* \\ | \\ b^* \end{bmatrix} = \frac{1}{2}\begin{bmatrix} | \\ \mathbf{w}^* \\ | \\ b^* \end{bmatrix}^T \begin{bmatrix} | \\ \mathbf{w}^* \\ | \\ 0 \end{bmatrix} + 0$$

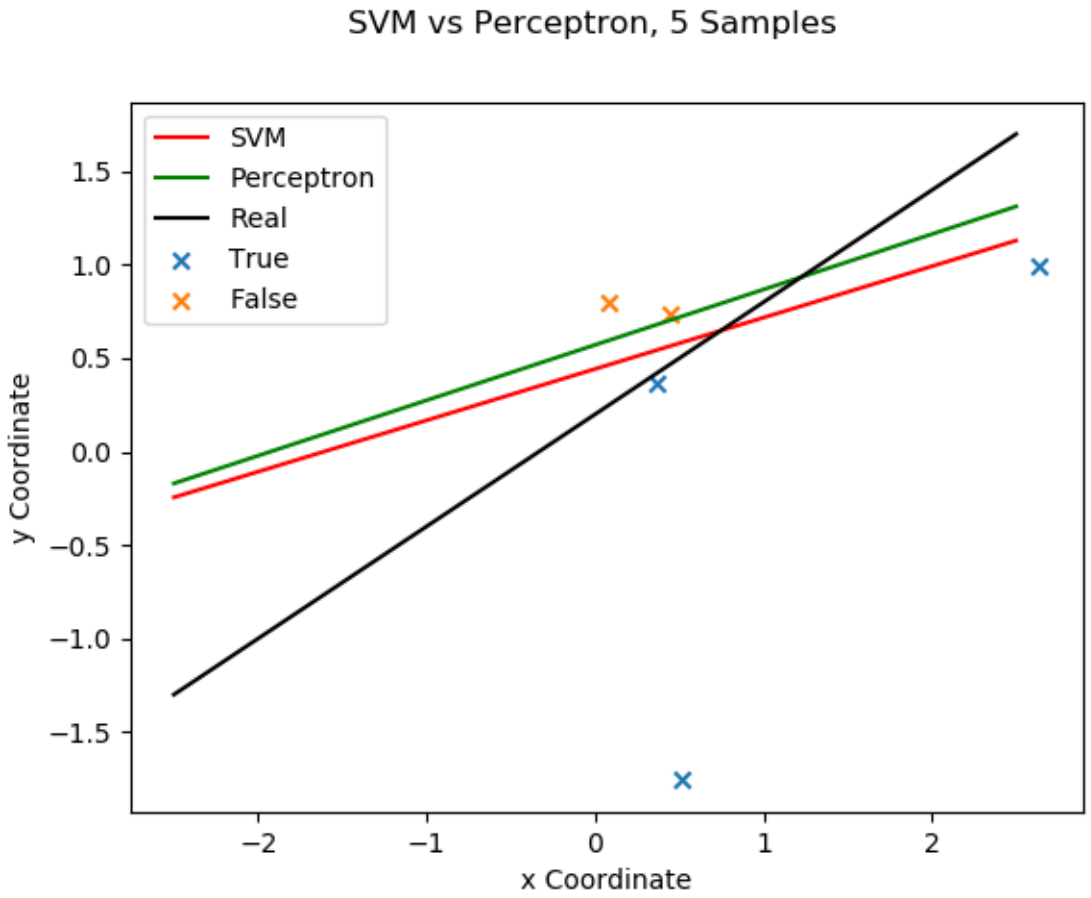
$$= \frac{1}{2}\|\mathbf{w}^*\|$$

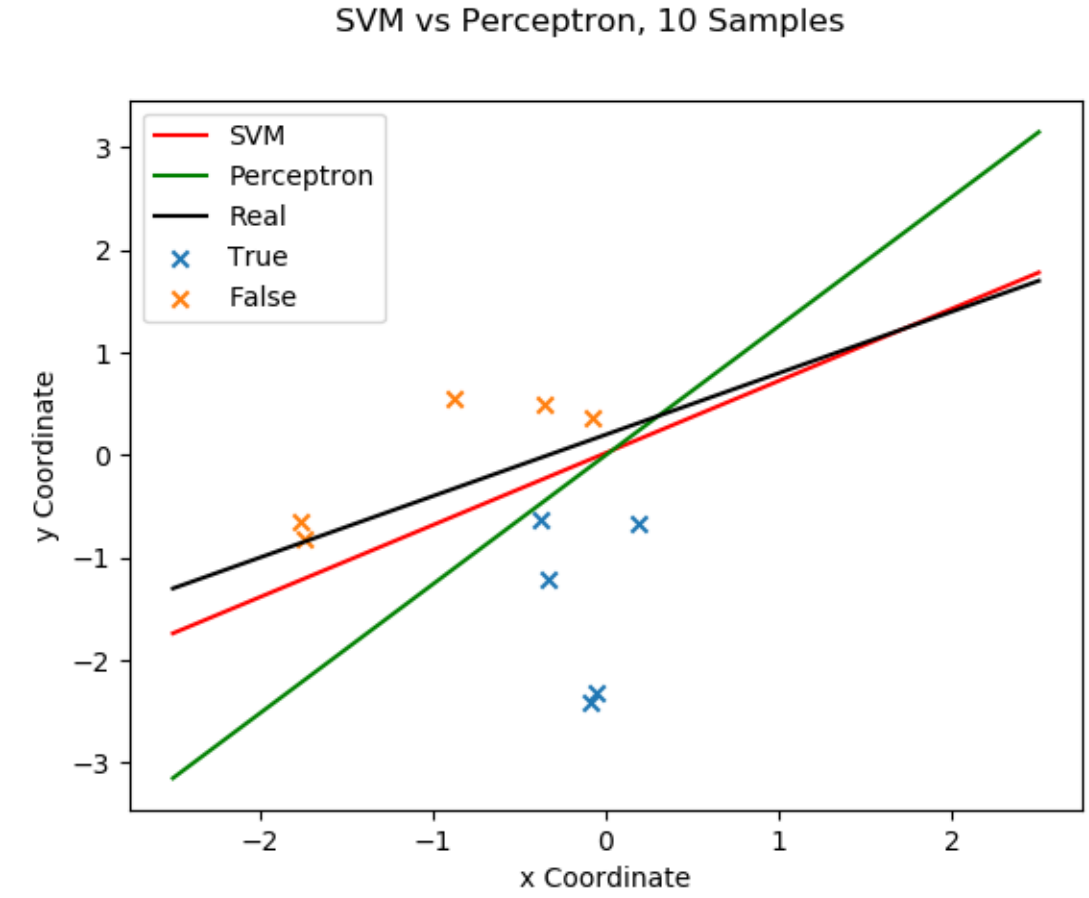
  - Since $\mathbf{w}^*$ optimizes $\|\mathbf{v}\|$, it also optimizes $\frac{1}{2}\|\mathbf{w}^*\|$.
  - Moreover, the restriction from the original objective can be rewritten linearly as we've done: $\begin{bmatrix} - & -\mathbf{x}_1 & - & -1 \\ - & -\mathbf{x}_2 & - & -1 \\ & \vdots & & \\ - & -\mathbf{x}_m & - & -1 \end{bmatrix} \begin{bmatrix} | \\ \mathbf{w} \\ | \\ b \end{bmatrix} \begin{bmatrix} -1 \\ -1 \\ \vdots \\ -1 \end{bmatrix}$.

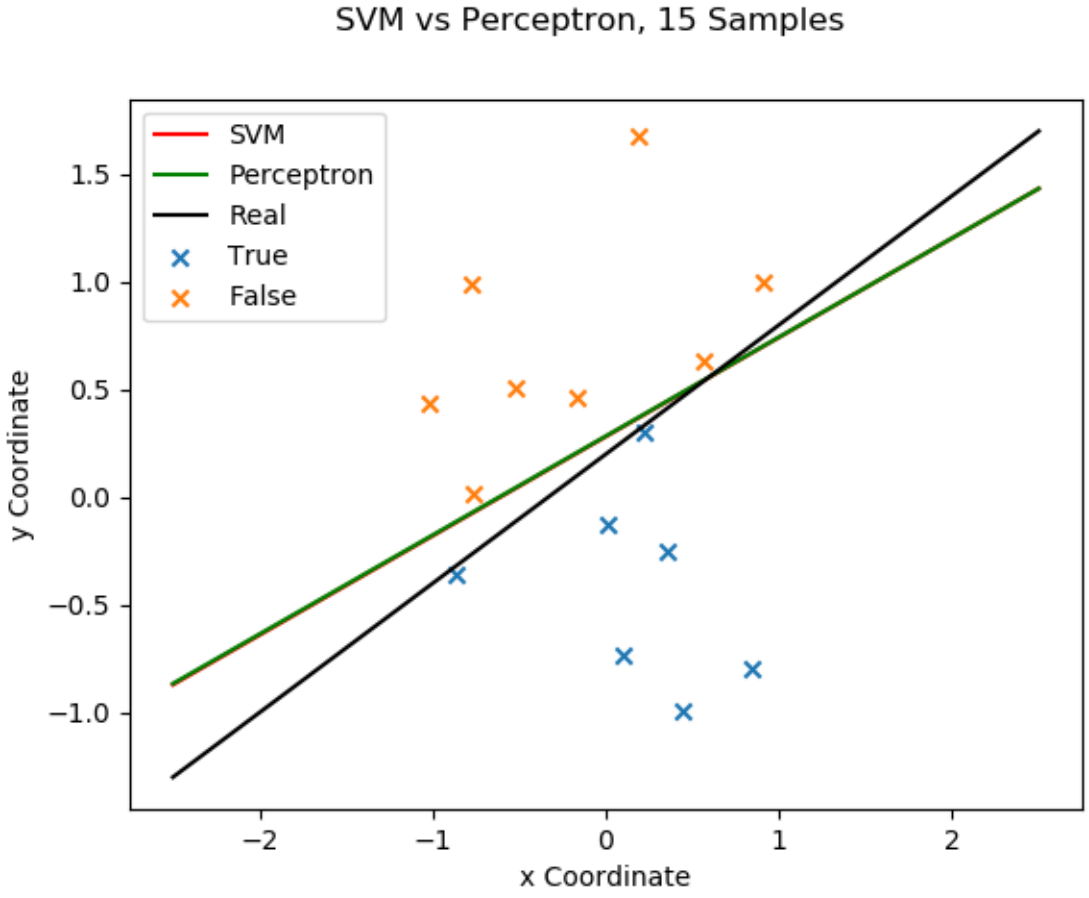
■

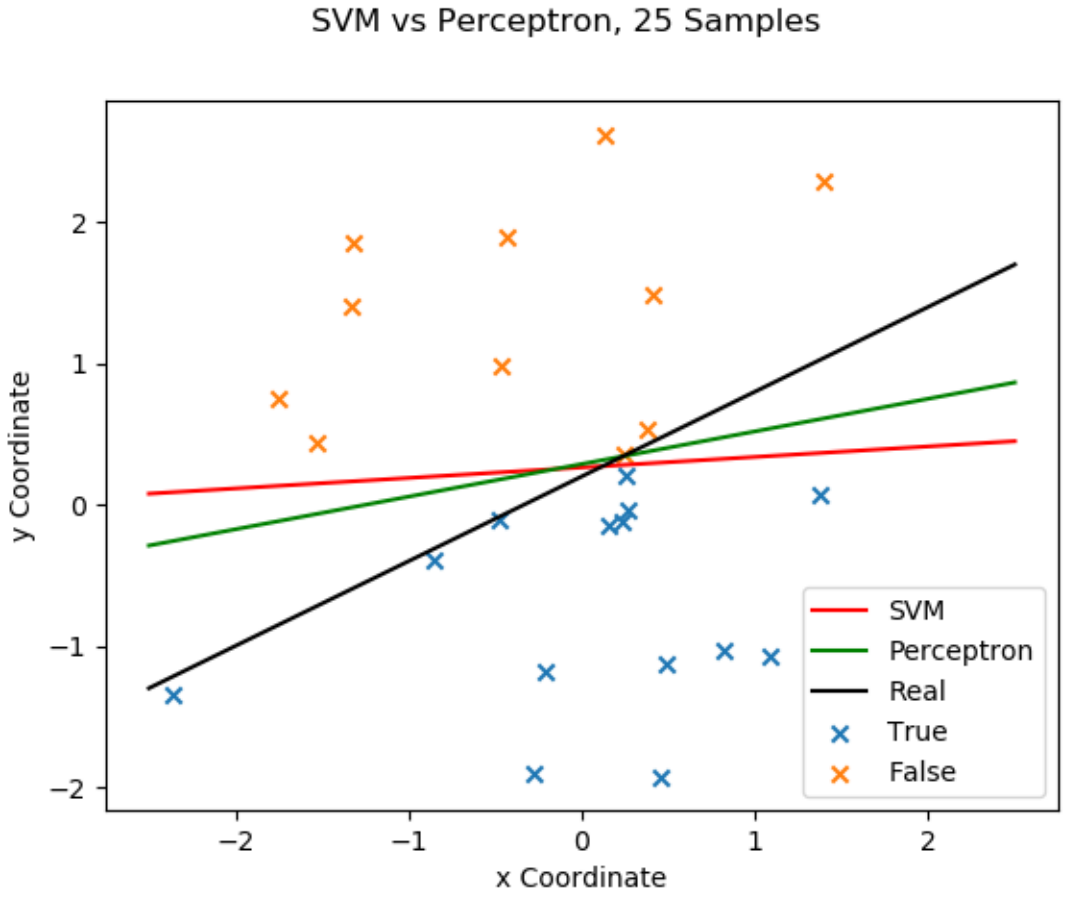
SVM and Generalization

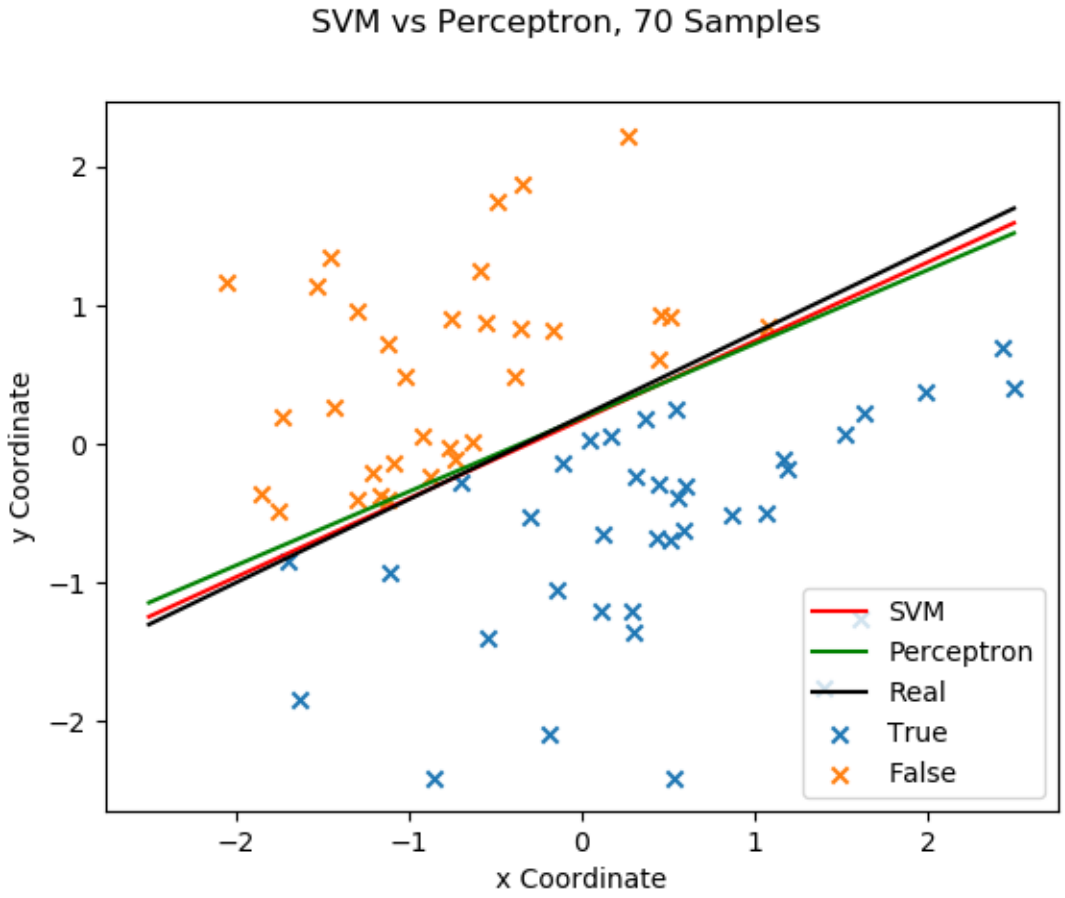
Q4




SVM vs Perceptron, 10 Samples
SVM
Perceptron
Real
True
False
y Coordinate
x Coordinate
3
2
1
0
−1
−2
−3
−2
−1
0
1
2

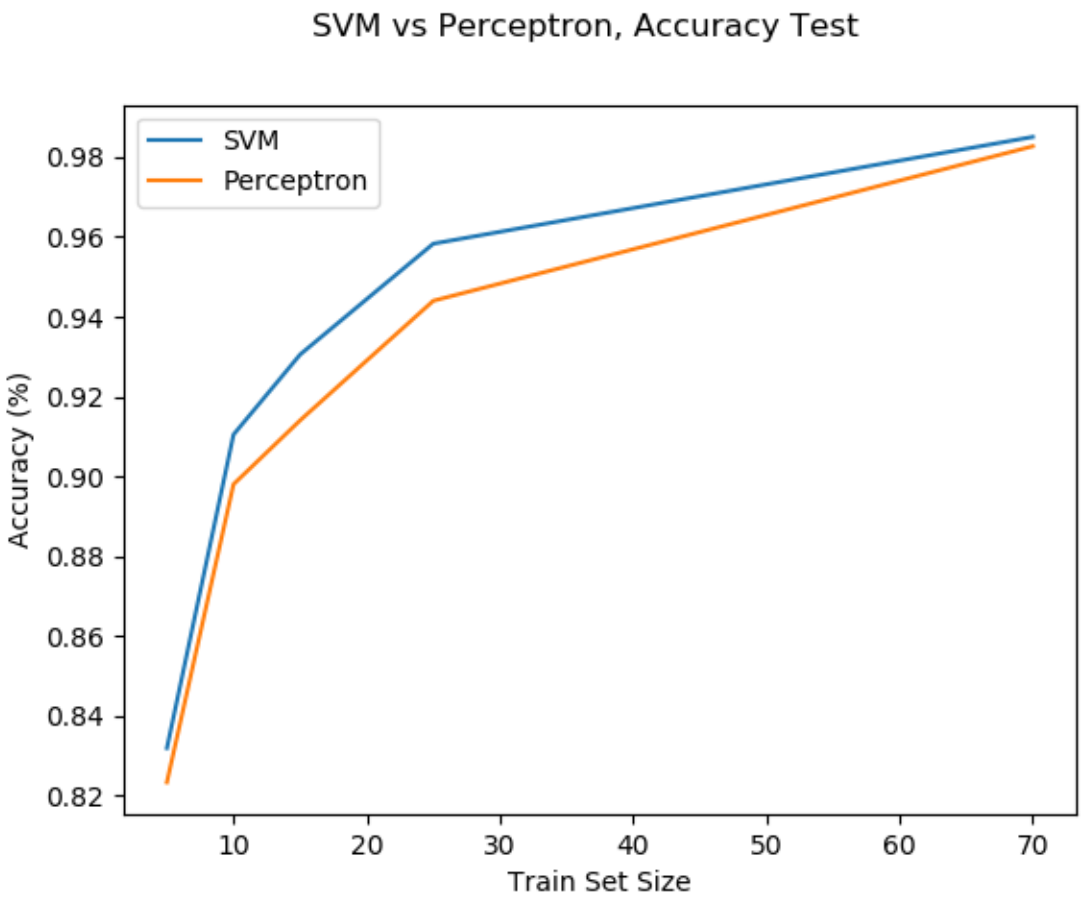

SVM vs Perceptron, 15 Samples
SVM
Perceptron
Real
True
False
y Coordinate
x Coordinate
1.5
1.0
0.5
0.0
−0.5
−1.0
−2
−1
0
1
2


SVM vs Perceptron, 25 Samples
y Coordinate
x Coordinate
SVM
Perceptron
Real
True
False


SVM vs Perceptron, 70 Samples
y Coordinate
x Coordinate
SVM
Perceptron
Real
True
False

Q5+6

- In the following graph we can see that the SVM did better than the Perceptron.
- This is mostly due to the fact that the SVM finds the separating line which has the largest margin, while the perceptron finds any separating line (the first one it finds).
- A bigger margin translates into a better generalizing line, hence the results.



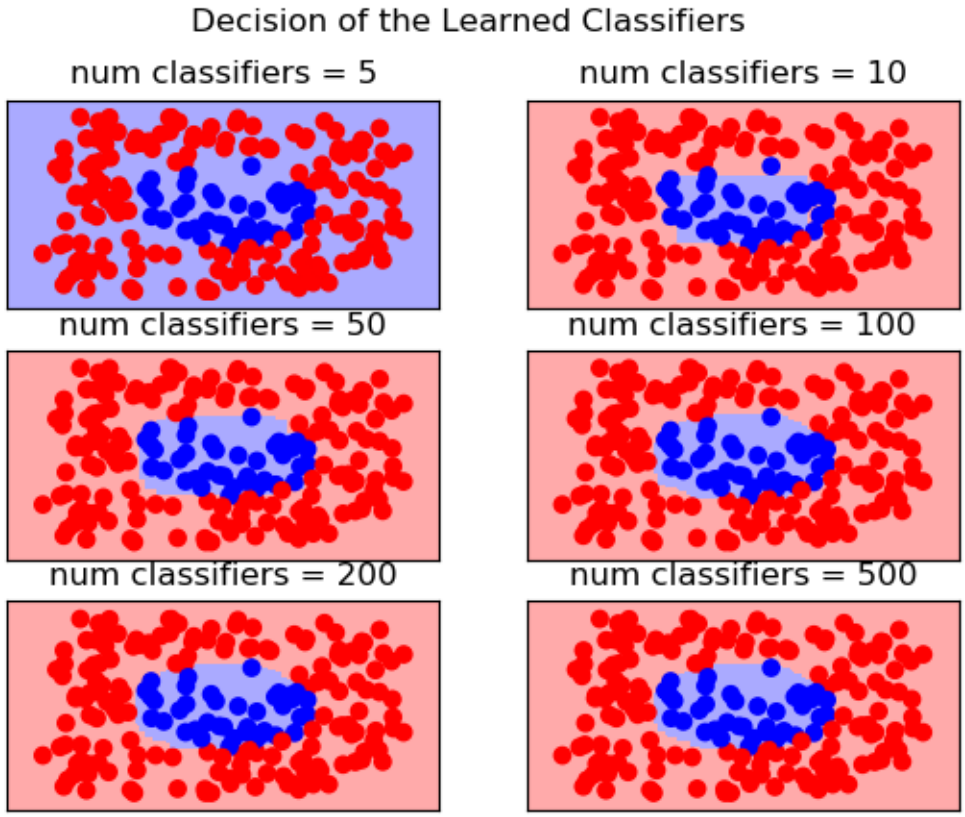
Separate the Inseparable - Adaboost

Q8

- The following is the graph of the error as a function of T, for both the train set and the test set.

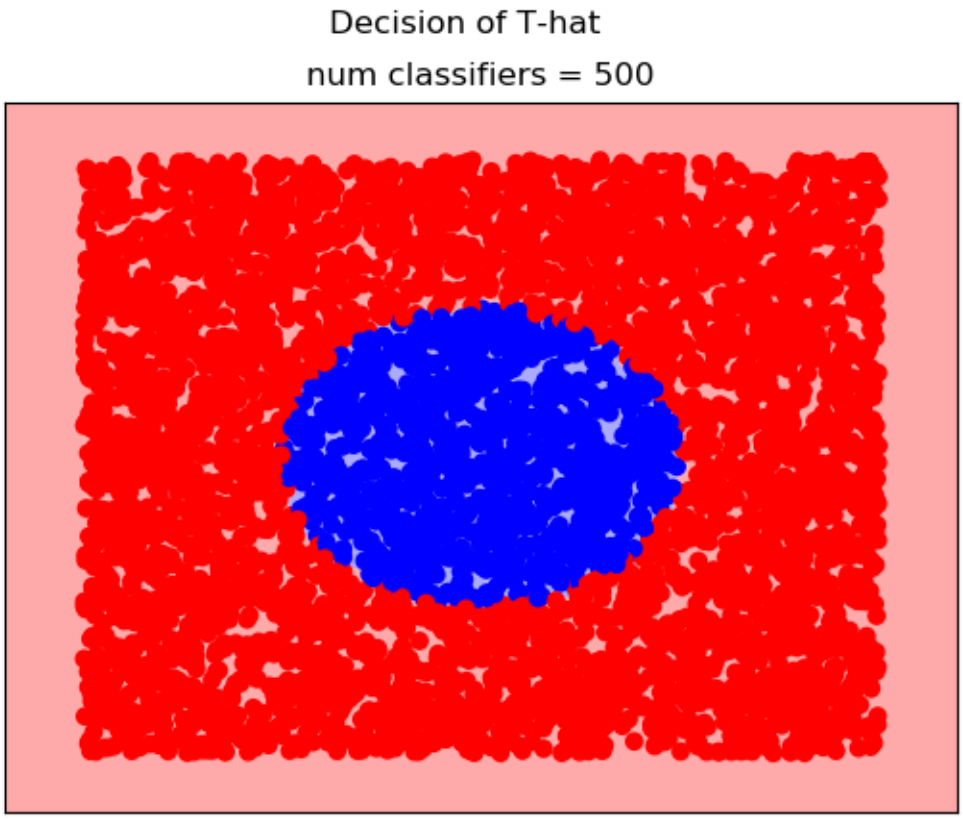

Train vs Test error, Adaboost
Train Error
Test Error
Error rate (%)
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
100
200
300
400
500
# of Hypotheses (T)

Q9


Decision of the Learned Classifiers
num classifiers = 5
num classifiers = 10
num classifiers = 50
num classifiers = 100
num classifiers = 200
num classifiers = 500

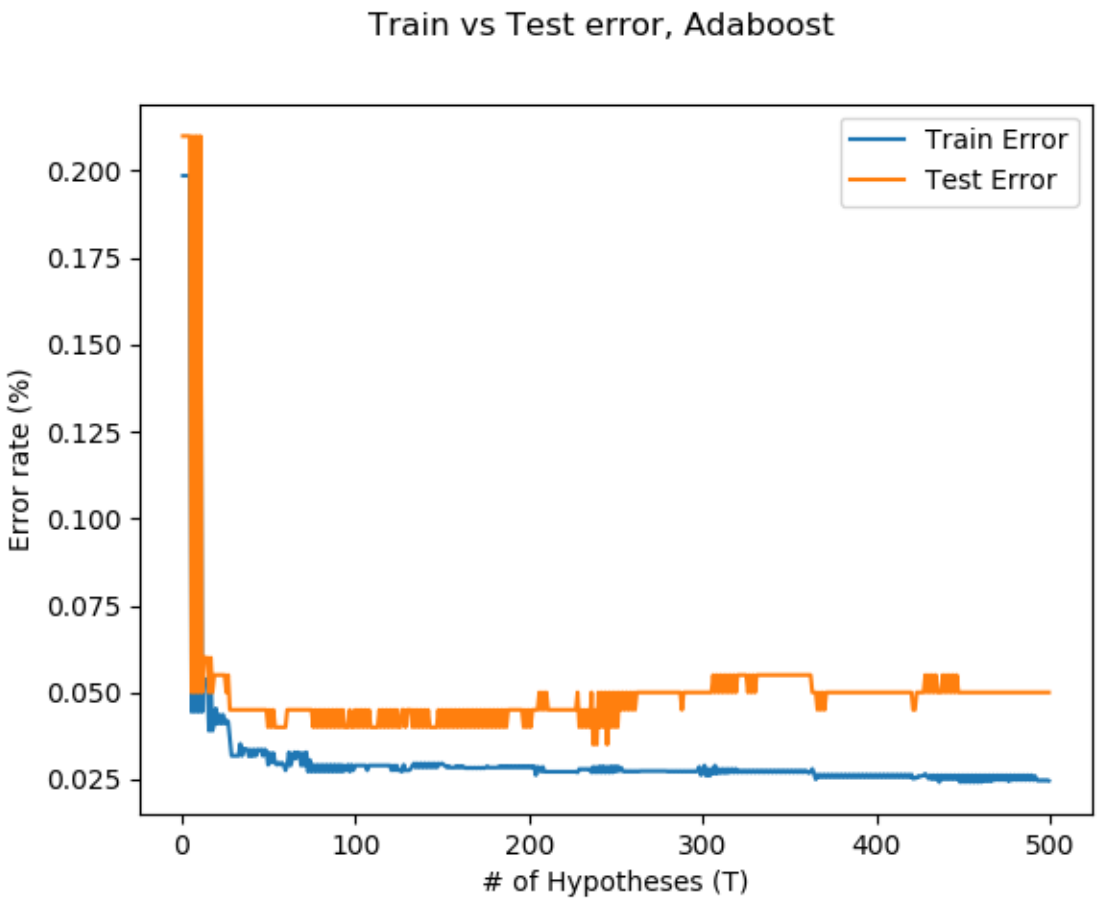
Q10

- As we can deduce from the graph in Q8, we see that $\hat{T}$ is equal to the largest T we took, which is 500.
- Its test error stabilizes at ~ 0.017.

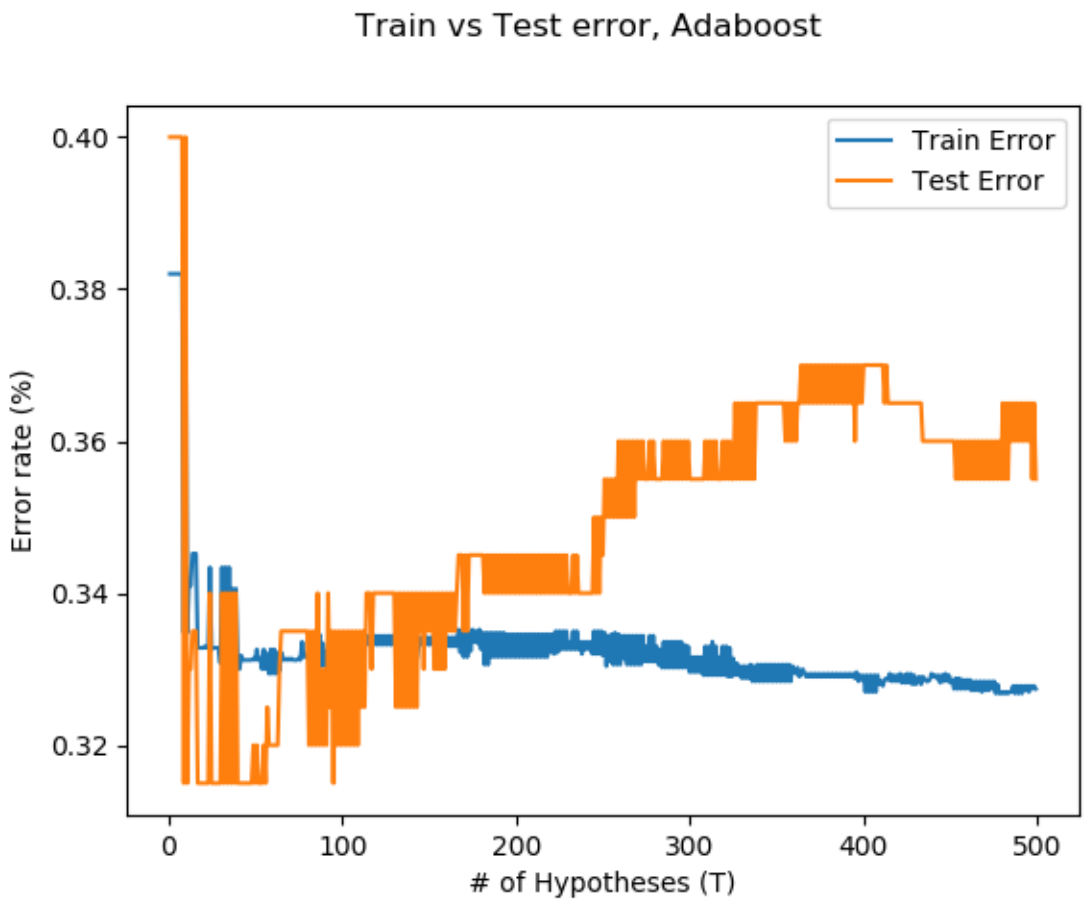


Q12

- $Q8$ with noise:
  - We can see that when noise was introduced, the test error curve changed from a monotonically decreasing curve, to a parabola-like curve with a minimum point.
  - This is due to what's called overfitting - when the hypotheses class becomes more complex, it starts to adjust to the bias (~noise), thus generalizing not as good.
  - In the case of Adaboost, the comlexity is controlled by the number of classifiers used in predicting new data - the X axis in these plots.
  - Noise = 0.01:

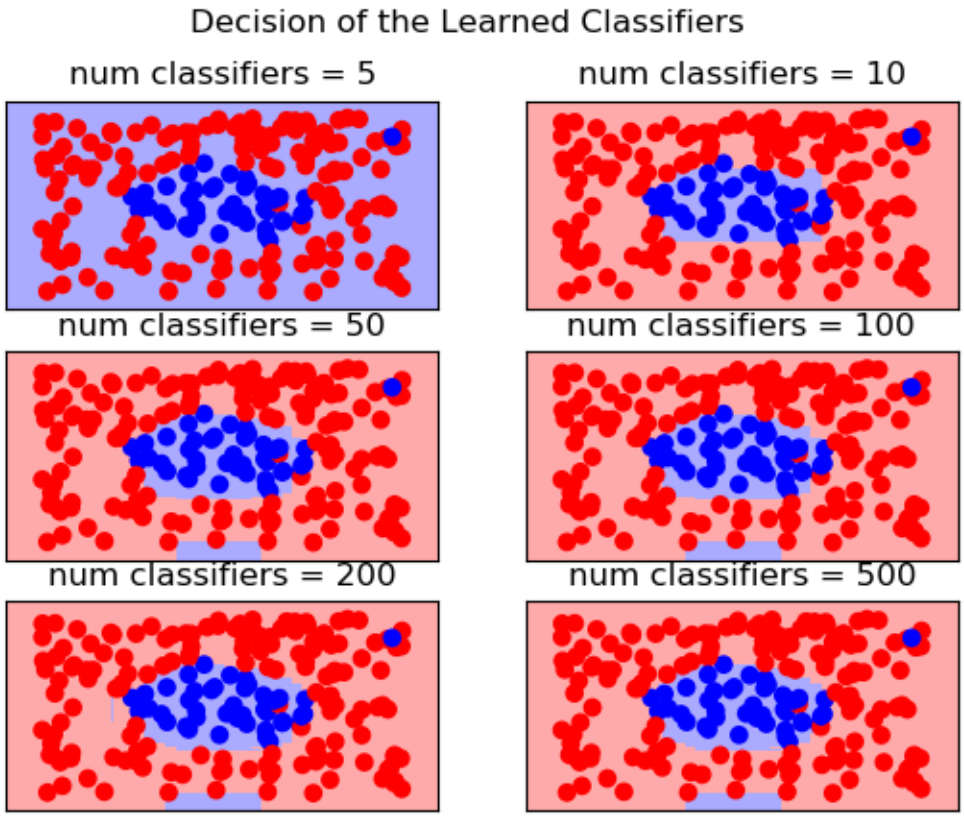


– Noise = 0.4:

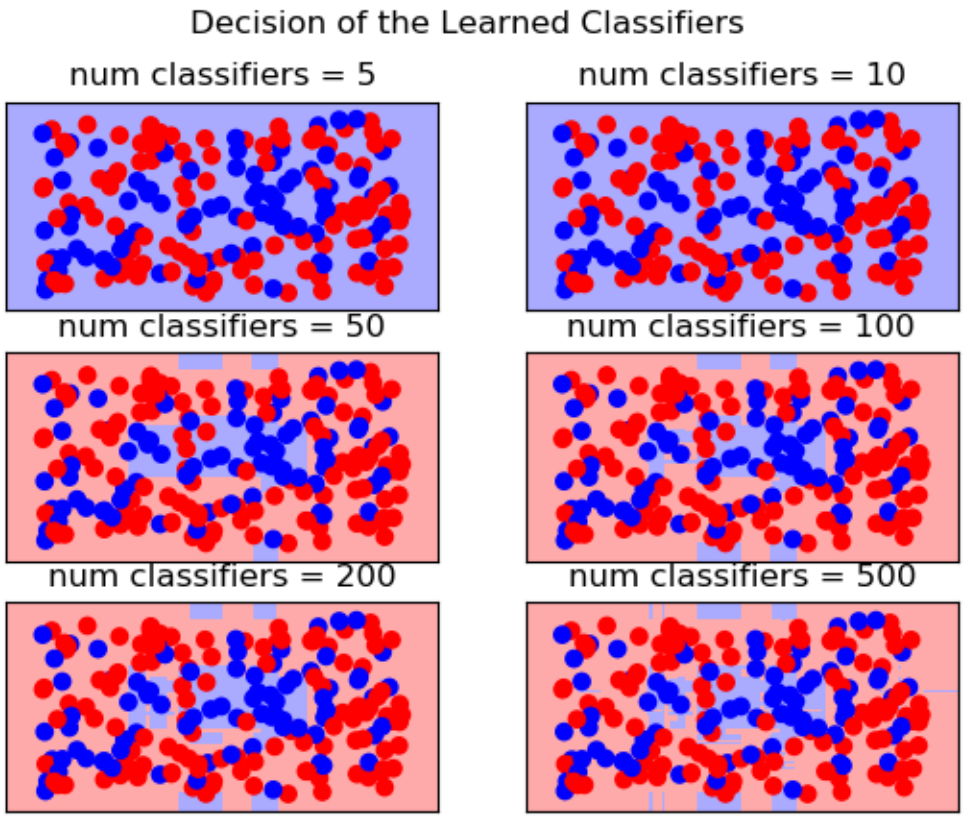


- $Q9$ with noise:

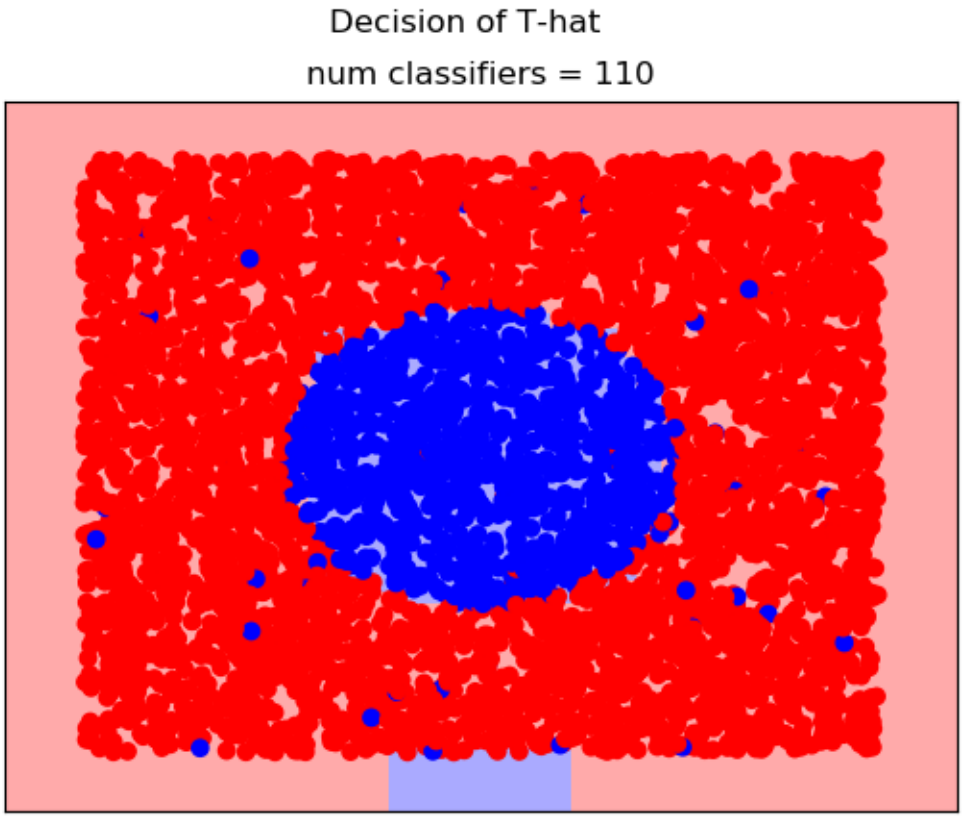
– Noise = 0.01:



– Noise = 0.4:



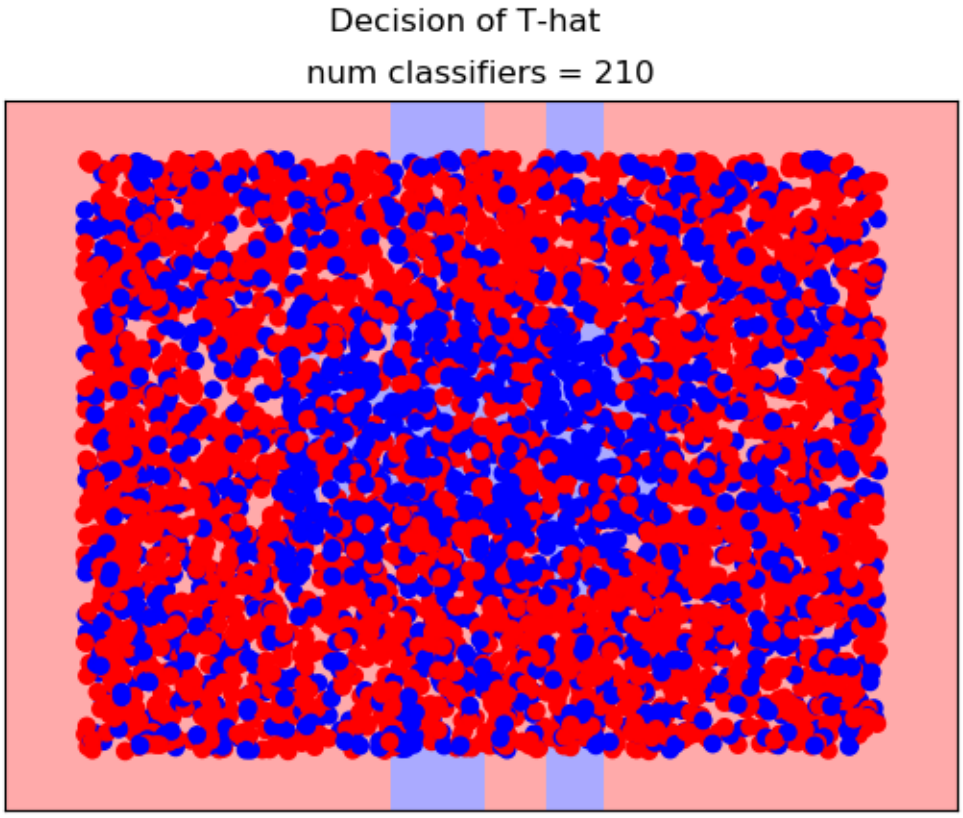
- $Q10$ with noise:
  - Here, we see that $\hat{T}$ is the one that hits the bias-variance tradeoff on spot.
  - Noise = 0.01:



  - Noise = 0.4:

Algorithm 1 Calculating $S(a,b)$ in $O(n)$

1. Initialize $sum \leftarrow I(a,b)$.
2. Update $sum \leftarrow sum + S(a-1,b)$.
3. Update $sum \leftarrow sum + S(a,b-1)$.
4. Update $sum \leftarrow sum - S(a-1,b-1)$.



Face Classification

Q13

- First of all, w.l.o.g assume (a,b) isn't an edge pixel (the other case can be handled with minor adjustments).
- Runtime analysis:
  - Stage 1: $O(1)$,
  - Stage 2: $O(1)$,
  - Stage 3: $O(1)$,
  - Stage 4: $O(1)$.
  - Overall, $O(1)$.
  - Since we do this for every entry in the image, we get $O(n)$, as required.

Q14

- **Algorithm:**
- **Rational:**
  - Similar to the Inclusion-exclusion principle, we add and subtract areas of the integral image that we're not added or added twice.

Algorithm 2 Finding Sum of Square in $O(1)$

Input:

- Integral image denoted by S,
- A square represented by four points, $P_\Delta = (a_\Delta, b_\Delta)$ for $\Delta \in \{LU, LD, RU, RD\}$ corresponding to their orientation (LU = Left-Up etc.).

Algorithm:

1. Initialize $sum \leftarrow S(P_{RD})$.
2. Update $sum \leftarrow sum - S(a_{LD}, b_{LD} - 1)$.
3. Update $sum \leftarrow sum - S(a_{RU} - 1, b_{RU})$.
4. Update $sum \leftarrow sum + S(a_{LU} - 1, b_{LU} - 1)$.
5. *return sum*.

Algorithm 3 Calculating a Haar Feature in $O(1)$

1. Calculate sum of all squares.
2. Add up the white squares, subtract the black ones.
3. *return* the result.

Q15

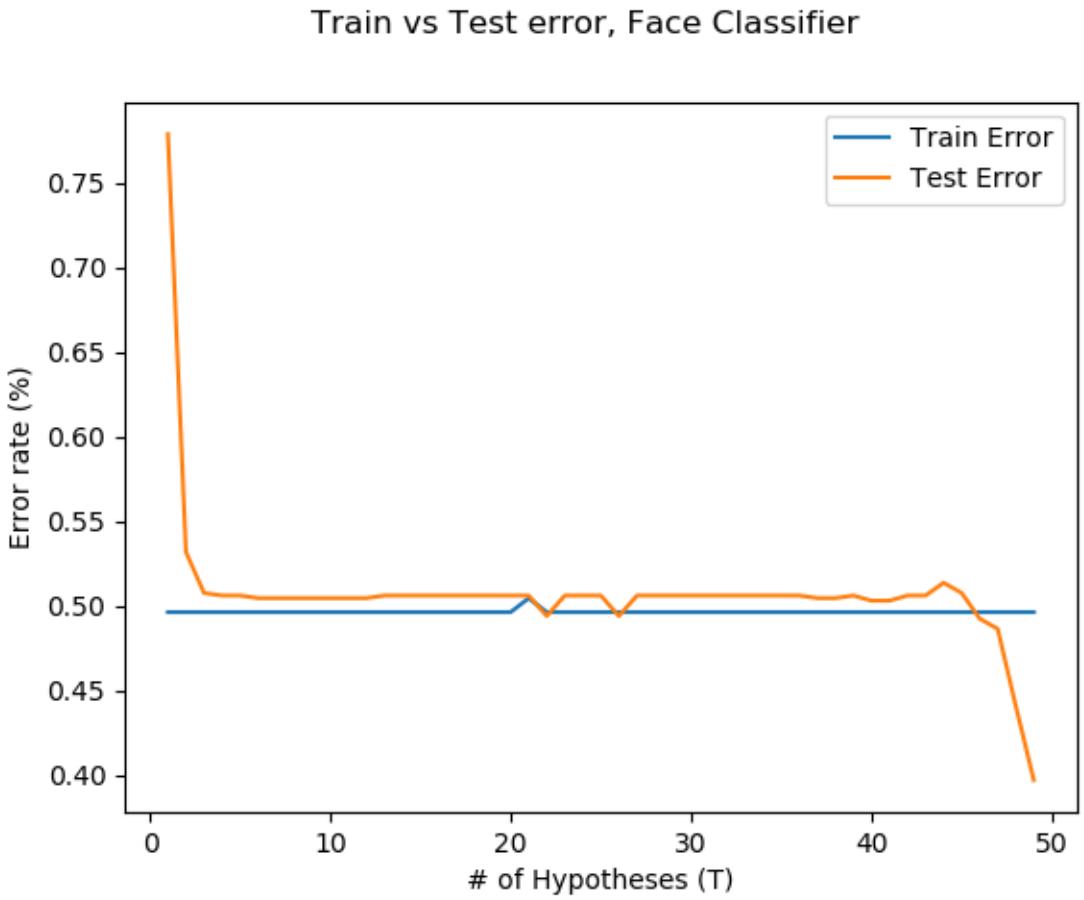
- **Algorithm:**
- **Runtime analysis:**
  - Stage 1, 2 and 3 all take $O(1)$ time, thus the overall procedure takes $O(1)$. ■

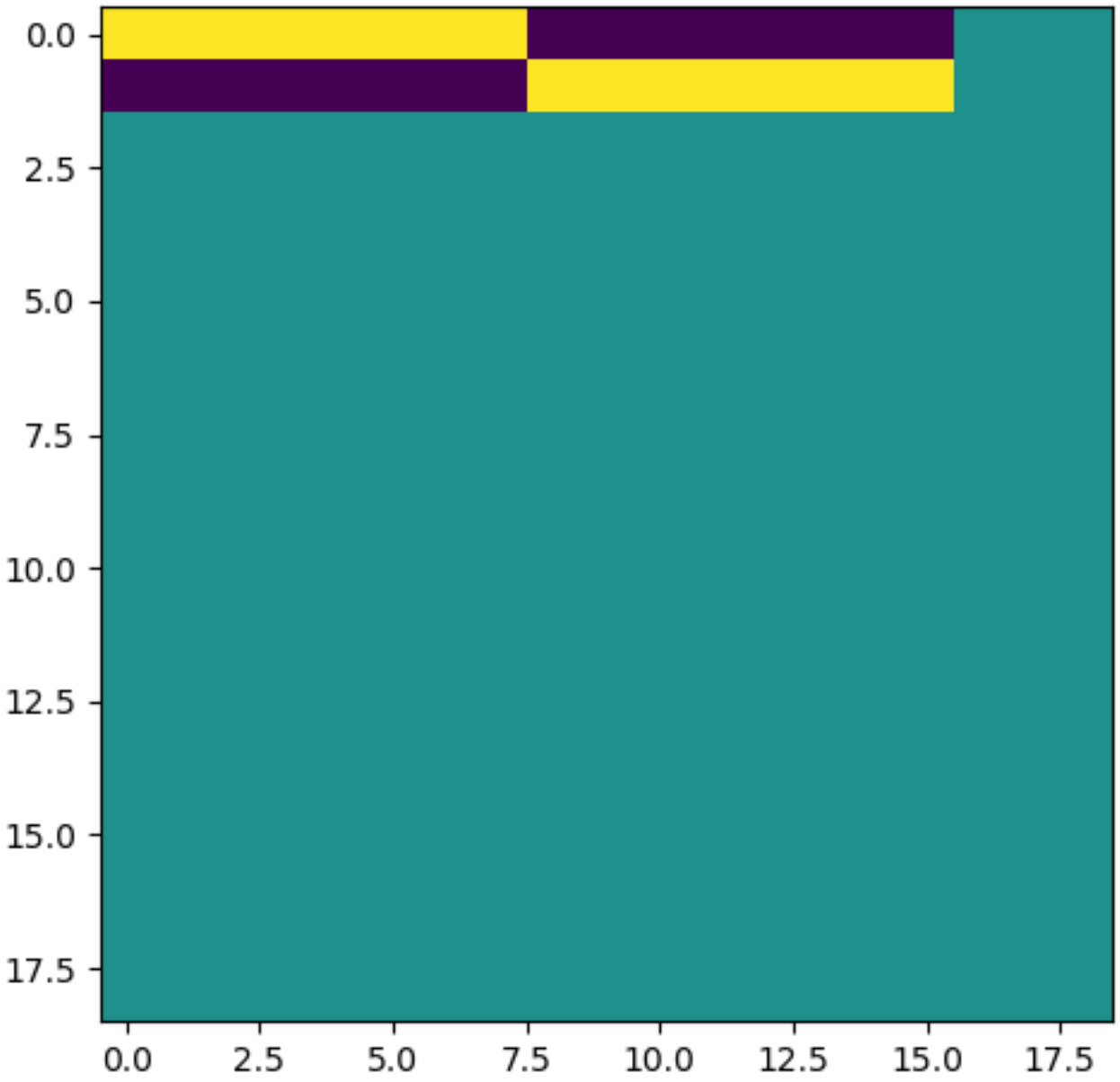
Q17

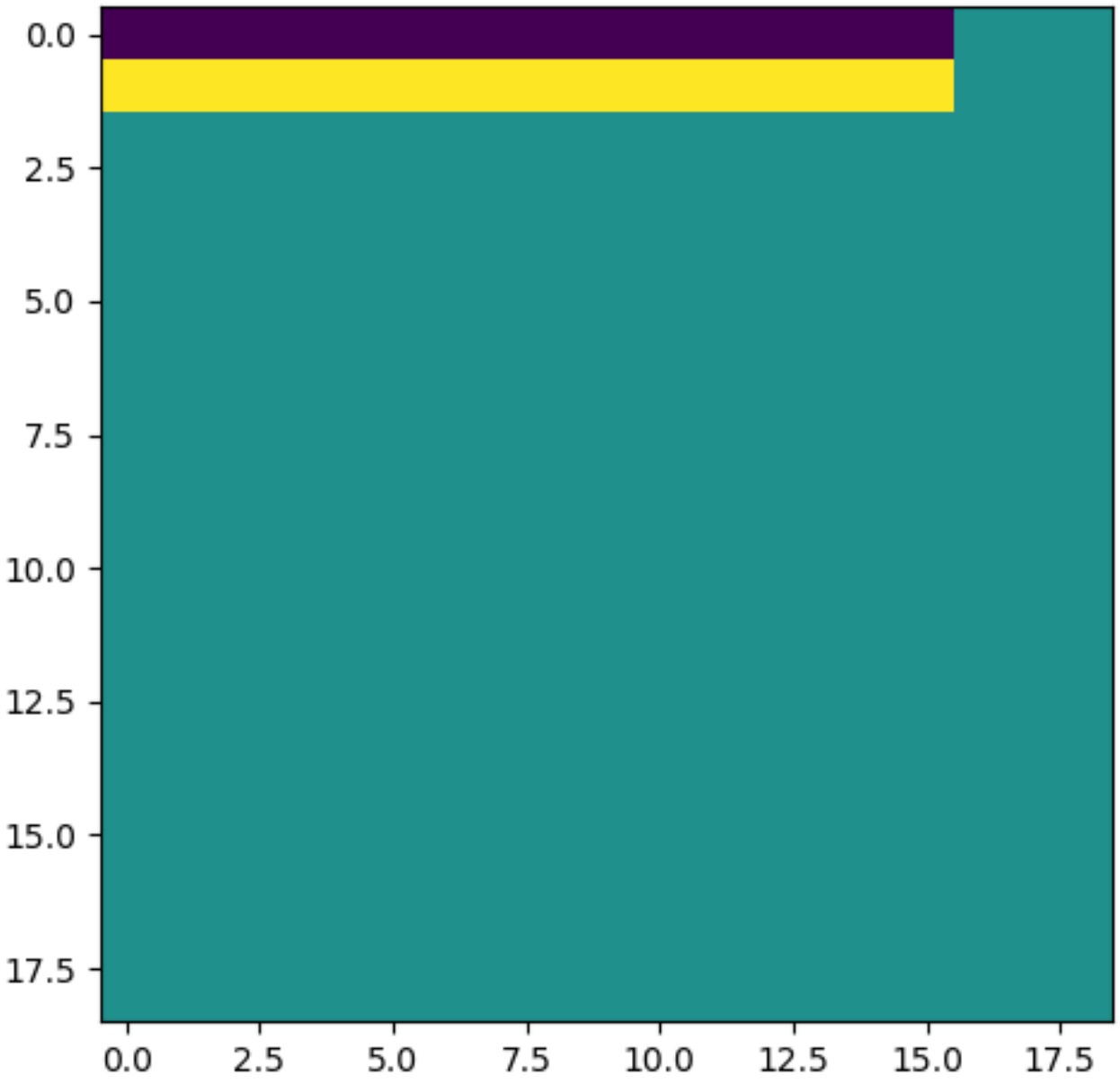
Implemented and ran it, but the result aren't good...

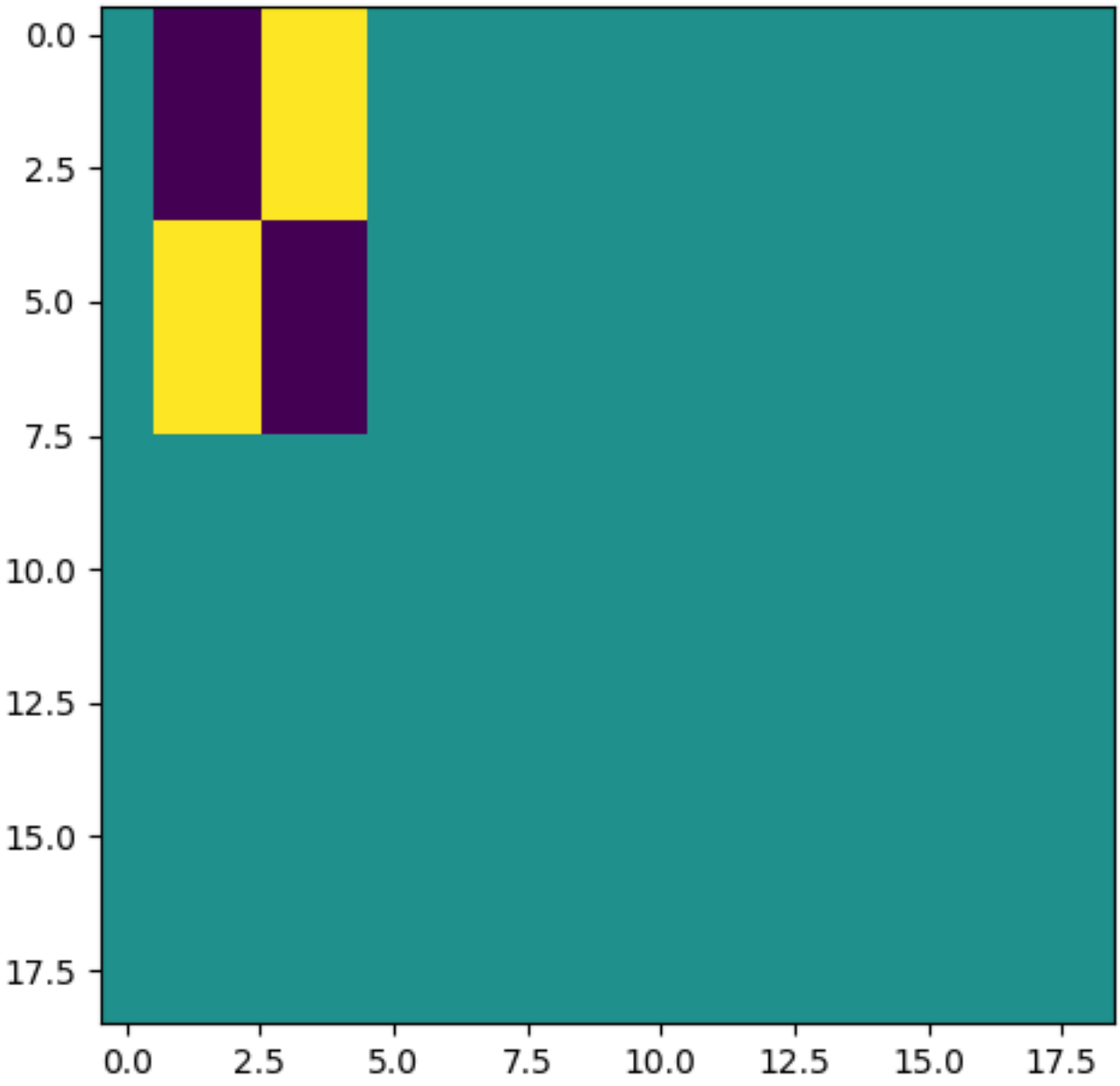
During the training, I plotted the optimal Haar features, and they all seem to have an 'up' value of 0 (= they're all locked to the top border of the image).

Couldn't find the bit of code that caused it though.


Train vs Test error, Face Classifier
Train Error
Test Error
0.75
0.70
0.65
0.60
0.55
0.50
0.45
0.40
Error rate (%)
0
10
20
30
40
50
# of Hypotheses (T)


0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5