Assignment 2: Introduction to Big Data

Submitted by: Pan Eyal 208722058 ; Yotam Lifschytz 209579077

1) Code added in attached zip file.

2) We got the following graph (a & b):

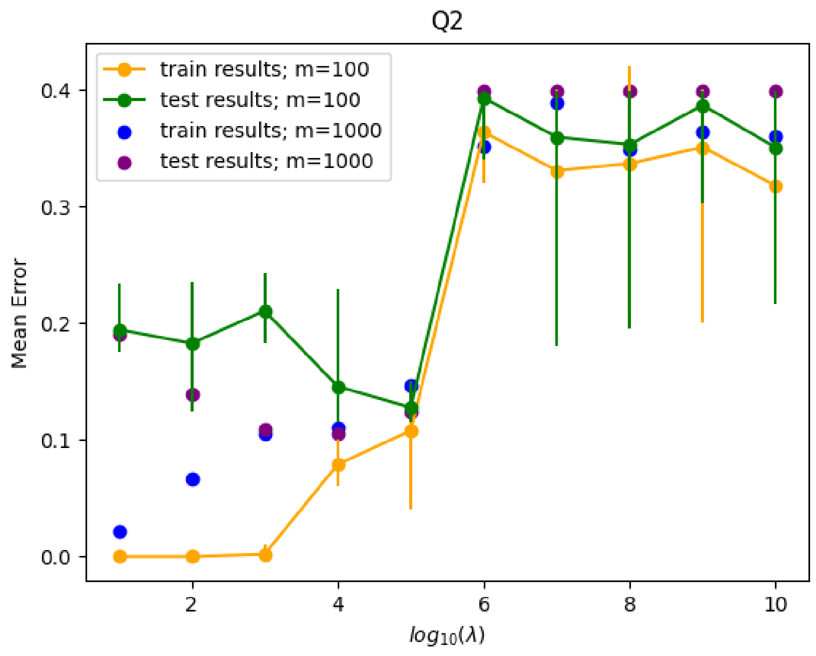

Figure 1

c)

- Train error: Based on what was taught in class, we would expect for the error measured on the training sample (“train results” as shown in fig. 1) to grow or stay approximately the same, as the size of the sample grows. This is because if the distribution is non separable, we may still choose a sample that is separable. As the size of the sample grows, it becomes more representative of the distribution and of the measure of “inseparableness” of the distribution. We observe this in fig. 1.
- Test error: We would expect the larger sample to have a lower training error due to the sample being more representative of the distribution, thus having a lower estimation error (this also can be seen in fig. 1).

- $\underline{\lambda:}$ Signifies the tradeoff between the estimation error and the approximation error. We want w with as large a margin as possible considering constraints (as we believe this represents the distribution most likely), but also minimize the hinge-loss (that benefits from a small margin), so we have a trade-off.
  For smaller λ, most of the “weight” of the error lies terms, thus we expect SVM to minimize the margin, resulting in a low training error but a high testing error (again, as a small margin solution is not a representative solution on the distribution). As λ increases, we expect these errors to approach each-other, and coalesce at a “sweet-spot” where the tradeoff is optimal (this is where the test error will be lowest). For still larger λ, we put more weight than “needed” on the norm term, thus minimizing the norm more than needed – this will result in a larger “penalized region” where even correctly labeled examples are considered errors as they are too close to the margin (this is due to the definition of hinge-loss).
  We see in fig.1 that we approach the optimal trade-off at $\lambda \approx 10^4$, and soon after we have an immediate and significant increase in the error, after which the error “stabilizes” and fluctuates slightly only due to statistical reasons (samples are randomly chosen anew for each λ).

3) Code added in attached zip file.

4) a) Points in the training set:

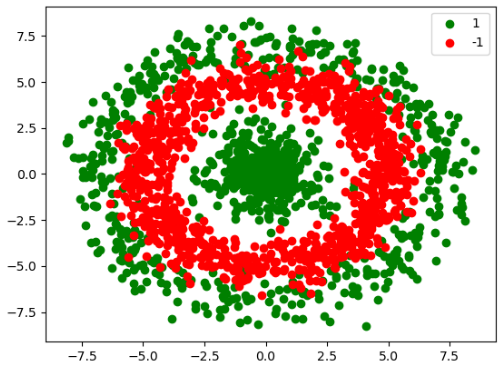

Figure 2

As we can see from the scattering of the data, a linear separator wouldn't be able to separate this data in any meaningful way. (There will always be '-1' and '1' labels from both sides of the separator)

By using a kernel soft SVM, and specifically gaussian kernel, we could resolve this issue and find a better fitting separator.

b) After running our implementation of soft-svm-rbf on the data, the cross-validation results of 9 parameter pairs are as follows:

	$\lambda = 1$	$\lambda = 10$	$\lambda = 100$
$\sigma = 0.01$	0.0805	0.0805	0.0805
$\sigma = 0.5$	0.0685	0.0685	0.0685
$\sigma = 1$	0.0880	0.0880	0.0880

As we can notice, lambda values do not affect the error. Therefor the cross-validation algorithm chose the first lambda value (=1) and returned the pair: $(\lambda = 1, \sigma = 0.5)$ as the best values.

Running the algorithm with this pair on the entire set achieved an error of 0.045.

When running our implementation of soft-svm (no kernel), we received the following results:

$\lambda = 1$	$\lambda = 10$	$\lambda = 100$
0.494	0.494	0.494

The chosen lambda value was 1. Running the algorithm with it on the entire set achieved an error of 0.51

c) RBF clearly achieved a better validation error as we expected. The received error from the soft-svm algorithm was very close to 50% and wouldn’t be able to predict any meaningful label for given data.
When the data samples distribute the same as the gaussian distribution, the RBF SVM will have a better validation error.
If the data on the other hand is separable (or close to separable) in the original feature space, transforming it with any non-linear Kernel will result in worse error than the original error of a linear separator in the original feature space.

d)

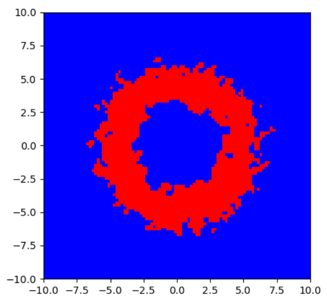

Figure 5: σ = 0.01

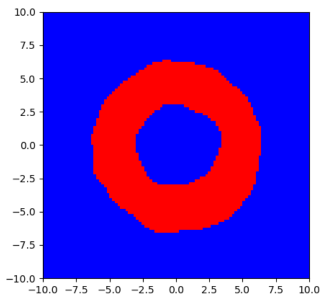

Figure 4: σ = 0.5

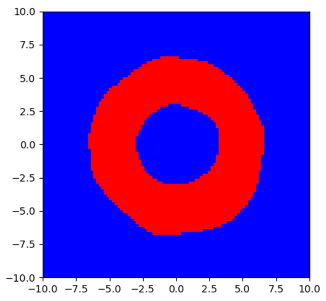

Figure 3: σ = 1.0

e) As we learned in class, small sigma values can have a very fine-tuned boundary as we can see from the first example when sigma=0.01
This results in larger estimation error (Overfitting).
As sigma grows, the function become smoother.
This results in larger approximation error (But supposedly generalizes better to the distribution).

5)

a) We now have the problem:

$$Minimize_{w\in\mathbb{R}^d}\ \lambda\|w\|^2 + \sum_{i=1}^{m}\xi_i^2$$

And we have the same constraints as before, as we choose to define ξ_i the same as in the standard fashion:

$$\forall i \quad \xi_i \geq 0$$

$$\forall i \quad \xi_i \geq 1 - y_i\langle w, x_i\rangle$$

b) We have:

$$u = \vec{0}$$

$$H = 2\cdot\begin{pmatrix}\lambda\hat{I}_{dd} & \hat{0}_{mm}\\ \hat{0}_{mm} & \hat{I}_{mm}\end{pmatrix}$$

- And this is the key part in this new version – H is defined s.t also the ξ_i take part in the nonlinear term.

$$A = \begin{pmatrix}\hat{B} & \hat{I}_{mm}\\ \hat{0}_{md} & \hat{I}_{mm}\end{pmatrix}$$

$$v = \left(\underbrace{1,\ldots,1}_{d},\underbrace{0,\ldots,0}_{m}\right)$$

Where:

$$\hat{B}_{ij} = y_i x(i)_j$$

And:

$$\hat{I}_{a,b} \equiv identity\ matrix\ of\ size\ a\times b$$

$$\hat{0}_{a,b} \equiv zero\ matrix\ of\ size\ a\times b$$

Note: if the quadratic program uses $Az \leq v$ instead of $Az \geq v$ we will use the same definitions defined above, but with $A \rightarrow -A, v \rightarrow -v$.

6) We will bring $Minimize_{w\in\mathbb{R}^d} r\|w\|_2^4 + \sum_{i=1}^{m} exp^{|<w,x_i> - y_i|}$ to the form:

$$Minimize_{w\in\mathbb{R}^d} f(< w, \psi(x_1) >, \dots, < w, \psi(x_m) >) + R(\|w\|)$$

By setting:

$$f(< w, \psi(x_1) >, \dots, < w, \psi(x_m) >) = \sum_{i=1}^{m} exp^{|<w,x_i> - y_i|}$$

Where we have:

$$\psi(x_i) \equiv x_i$$

And:

$$R(x) = r * x^4$$

To maintain R(x) property of a monotonic non-decreasing function, we must demand that r ≥ 0. Therefor: $r \in [0, \infty)$

Proof of correctness is trivial, as $x \to x^4$ is a monotonic non decreasing function, $x \to rx$ where r ≥ 0 is a monotonic non decreasing function, and composition conserves this property. We can see that the set of r values is maximal, as the only way to enlarge it is to allow negative values, which would make $x \to rx$ a decreasing function for $x \in \mathbb{R}^+$ which is the domain of R. Thus, this is the exact set of values.

7)

a) K is a kernel function iff $K(x, x') = \langle \psi(x), \psi(x') \rangle$ for some $\psi(z)$.
Let's assume by contradiction $K(x, x') = -x(1)x'(1)$ is a kernel function, and let us consider the case of $x = x' \Rightarrow$

$$K(x, x) = \langle \psi(x), \psi(x) \rangle = \|\psi(x)\|^2 = -x(1)^2$$

For any real valued x with $x(1) \neq 0$, $-x(1)^2 < 0 \Rightarrow \|\psi(x)\|^2 < 0$.
This contradicts the definition of the norm; thus, we have a contradiction. K is not a kernel function.

b) Let's assume by contradiction $K(x,x') = \big(x(1)+x(2)\big)\big(x'(3)+x'(4)\big)$ is a kernel function.

Thus:

$$K(x,x') = \langle \psi(x), \psi(x') \rangle = x(1)x'(3) + x(1)x'(4) + x(2)x'(3) + x(2)x'(4)$$

We have, by definition:

$$\langle \psi(x), \psi(x') \rangle \equiv \sum_i \big(\psi(x)\big)_i \cdot \big(\psi(x')\big)_i$$

From this, we must deduce that (w.l.o.g):

$$\psi(x) = \big(x(1), x(1), x(2), x(2)\big)$$
$$\psi(x') = \big(x'(3), x'(4), x'(3), x'(4)\big)$$

Or some permutation of these values that conserves the relative ordering, so that the terms in the sum stay the same as above. In any case, we can easily see the contradiction in the definition of ψ if we look at the case $x = x'$, where we have the definitions:

$$\psi(x) = \big(x(1), x(1), x(2), x(2)\big)$$
$$\psi(x) = \big(x(3), x(4), x(3), x(4)\big)$$

Where this cannot generalize for all x, as for these definitions to not contradict we must have $x(1) = x(3) = x(4) = x(2)$ which is not the general example.

c) <u>The original Perceptron algorithm:</u>

```
i.   Input: A separable training sample $S = \{(\vec{x}_1, y_1), \dots, (\vec{x}_m, y_m)\}$
ii.  Set: $t = 1$ ,  $\vec{w}(t=1) = \left(\underbrace{0, 0, \dots, 0}_{d}\right)$
iii. $While(\exists i \rightarrow y_i \langle \vec{w}(t), \vec{x}_i \rangle < 0)$:
        1. $\vec{w}(t+1) = \vec{w}(t) + y_i \vec{x}_i$
        2. $t = t + 1$
iv.  Return $w$
```

The “Kernel Perceptron” algorithm:

* *Note: We remember that here, for some x_i we would like to have:*

$$\langle w, \psi(x_i)\rangle = \sum_{j=1}^{m} \alpha(j)\langle \psi(x_j), \psi(x_i)\rangle = \sum_{j=1}^{m} \alpha(j) G_{ji} = \langle \alpha, G[i]\rangle$$

```
i.   Input: The Gram matrix $\hat{G}$  $s.t$  $\hat{G}_{ij} = K(x_i, x_j)$,
            label set $Y = \{y_1, \dots, y_m\}$
ii.  Set: $t = 1$ ,  $\alpha(t = 1) = \left(\underbrace{0, 0, \dots, 0}_{m}\right)$
iii. $While(\exists i \to y_i\langle \alpha, G[i]\rangle < 0)$:
        1. $\alpha(t+1) = \alpha(t) + y_i G[i]$
        2. $t = t + 1$
iv.  Return $\alpha$
```

Why does this update rule work?

We can see that:

$$y_i\langle \alpha(t+1), G[i]\rangle = y_i\langle \alpha(t) + y_i G[i], G[i]\rangle =$$

$$y_i\langle \alpha(t), G[i]\rangle + y_i^2\langle G[i], G[i]\rangle = y_i\langle \alpha(t), G[i]\rangle + \|G[i]\|^2 \geq y_i\langle \alpha(t), G[i]\rangle$$

So, we always make $y_i\langle \alpha, G[i]\rangle$ larger (same logic as original perceptron).