1 Paper Problems

1.

a)

ξ_i can take any positive value when it breaks into the margin.

b)

When ξ_i stays on or outside the margin it will be 0.

c)

We include the term in order to allow some examples to break into the margin. Meaning that we allow for incorrect guesses in order to allow us to use linear classifiers on non-linearly separable data. By minimizing this variable we can minimize the number of examples that break into the margin. If we were to exclude this case we do not allow any examples to break into the margin and we will not be able to linearly separate some data sets.

2.

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j - \sum_i \alpha_i \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C \\ & \sum_i \alpha_i y_i = 0 \end{aligned}$$

We derived this by starting with the primal and converting it to the Lagrange function by introducing the Lagrangian multipliers α_i and β_i and then taking the min max optimization.

From there we switched the max and min to switch to the dual and fixed the Lagrangian multipliers by solving the inner minimization. Which was done by setting the gradient to 0. After substituting those solved gradients back into the equation we eliminate the min and are just left with the maximization function. Which we can then take the negative of to convert it to a minimization and that gives the objective function listed above.

We then find that we never have to solve for β_i since it no longer occurs in the objective function so we can modify the first constraint so that α_i is less than C. Giving us the first constraint of the simplified dual.

The second constraint of the simplified dual, $\sum_i \alpha_i y_i = 0$, comes from solving $\frac{\delta L}{\delta b} = 0$.

3.

a)

α_i values that are equal to zero indicate that the i-th example lies outside the margin.

b)

Could be completely wrong since I cannot find anything in the lecture that talks about this but I'm assuming we could just find the examples in our support vectors such that $y_i(\sum_{j \in SV} \alpha_j y_j x_j^\top x_i + b^*) = 1$. As these examples will be the ones that lie exactly on the margin.

4.

Kernel trick allows us to elevate the data to a higher dimension in which it is linearly separable. We do so by using the kernel trick to efficiently calculate a higher dimensional mapping of the data that we can then train on. Then to get the predictions we just need to map the data into that dimension before applying the weight vector. The optimization problem is simply as follows

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j K(x_i, x_j) - \sum_i \alpha_i \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C \\ & \sum_i \alpha_i y_i = 0 \end{aligned}$$

5. This problem was a lot of work, now I know why we let the computer do this stuff... I'm really hoping I didn't make a stupid mistake in like the first part and get it all wrong and lose all of the points after like an hour or two of work.

Step 1

Example 1:

$$\begin{aligned} y_i * \mathbf{w}^\top \mathbf{x}_1 &= 0 \leq 1 \\ \nabla J &= [\mathbf{w}_0; 0] - \frac{1}{3} * 3 * 1 * [0.5, -1, 0.3, 1] = [-0.5, 1, -0.3, -1] \\ \mathbf{w} &= \mathbf{w} - 0.01 * [-0.5, 1, -0.3, -1] = [0.005, -0.01, 0.003, 0.01] \end{aligned}$$

Example 2:

$$\begin{aligned} y_i * \mathbf{w}^\top \mathbf{x}_1 &= -1 * [0.005, -0.01, 0.003, 0.01]^\top * [-1, -2, -2, 1] = -0.019 \leq 1 \\ \nabla J &= [0.005, -0.01, 0.003, 0] - \frac{1}{3} * 3 * -1 * [-1, -2 - 2, 1] \\ &= [-0.995, -2.01, -1.997, 1] \\ \mathbf{w} &= [0.005, -0.01, 0.003, 0.01] - 0.01 * [-0.995, -2.01, -1.997, 1] \\ &= [0.01495, 0.0101, 0.02297, 0.] \end{aligned}$$

Example 3:

$$\begin{aligned} y_i * \mathbf{w}^\top \mathbf{x}_1 &= 1 * [0.01495, 0.0101, 0.02297, 0]^\top * [1.5, 0.2, -2.5, 1] = -0.03298 \leq 1 \\ \nabla J &= [0.01495, 0.0101, 0.02297, 0] - \frac{1}{3} * 3 * 1 * [1.5, 0.2, -2.5, 1] \\ &= [-1.48505, -0.1899, 2.52297, -1] \\ \mathbf{w} &= [0.01495, 0.0101, 0.02297, 0] - 0.01 * [-1.48505, -0.1899, 2.52297, -1] \\ &= [-5.000e-05, 8.100e-03, 4.797e-02, -1.000e-02] \end{aligned}$$

Step 2

For brevity's sake I will be cutting out showing the middle steps, just know that I am doing them but I just didn't want to type them.

Example 1:

$$\begin{aligned} y_i * \mathbf{w}^\top \mathbf{x}_1 &= -0.00373 \leq 1 \\ \nabla J &= [\mathbf{w}; 0] - C * N * y_i \mathbf{x}_i \\ &= [-0.50005, 1.0081, -0.25203, -1] \\ \mathbf{w} &= \mathbf{w} - 0.005 * \nabla J \\ &= [0.00245025, 0.0030595, 0.04923015, -0.005] \end{aligned}$$

Example 2:

$$\begin{aligned} y_i * \mathbf{w}^\top \mathbf{x}_1 &= .112 \leq 1 \\ \nabla J &= [\mathbf{w}; 0] - C * N * y_i \mathbf{x}_i \\ &= [-0.99754975, -1.9969405, -1.95076985, 1] \\ \mathbf{w} &= \mathbf{w} - 0.005 * \nabla J \\ &= [0.007438, 0.0130442, 0.058984, -0.01] \end{aligned}$$

Example 3:

$$\begin{aligned} y_i * \mathbf{w}^\top \mathbf{x}_1 &= -0.1437 \leq 1 \\ \nabla J &= [\mathbf{w}; 0] - C * N * y_i \mathbf{x}_i \\ &= [-1.492562, -0.1869558, 2.558984, -1] \\ \mathbf{w} &= \mathbf{w} - 0.005 * \nabla J \\ &= [0.01490081, 0.01397898, 0.04618908, -0.005] \end{aligned}$$

Step 3

Example 1:

$$\begin{aligned} y_i * \mathbf{w}^\top \mathbf{x}_1 &= 0.00232 \leq 1 \\ \nabla J &= [\mathbf{w}; 0] - C * N * y_i \mathbf{x}_i \\ &= [-0.48509919, 1.01397898, -0.25381092, -1] \\ \mathbf{w} &= \mathbf{w} - 0.0025 * \nabla J \\ &= [0.01611356, 0.01144403, 0.04682361, -0.0025] \end{aligned}$$

Example 2:

$$\begin{aligned} y_i * \mathbf{w}^\top \mathbf{x}_1 &= .135 \leq 1 \\ \nabla J &= [\mathbf{w}; 0] - C * N * y_i \mathbf{x}_i \\ &= [-0.98388644, -1.98855597, -1.95317639, 1] \\ \mathbf{w} &= \mathbf{w} - 0.0025 * \nabla J \\ &= [0.01857327, 0.01641542, 0.05170655, -0.005] \end{aligned}$$

Example 3:

$$\begin{aligned} y_i * \mathbf{w}^\top \mathbf{x}_1 &= -0.103 \leq 1 \\ \nabla J &= [\mathbf{w}; 0] - C * N * y_i \mathbf{x}_i \\ &= [-1.48142673, -0.18358458, 2.55170655, -1] \\ \mathbf{w} &= \mathbf{w} - 0.0025 * \nabla J \\ &= [0.02227684, 0.01687439, 0.04532728, -0.0025] \end{aligned}$$

2 Practice Problems

1. c.

Seen below is a table of the training and test errors for the varying C values with the two learning rate schedules. I did not include the weights in this table because it would have made it a bit too messy in my opinion but the learned weights vary wildly for the differing values of C and the different learning schedules. As far as the training and test errors go the second learning schedule seems to benefit more from higher values of C than the first one does. In both cases the performance gets worse for C = 500/873 and then gets better.

	$\frac{\gamma_0}{1+\frac{\gamma_0}{\alpha}t}$		$\frac{\gamma_0}{1+t}$	
C	Training Error	Test Error	Training Error	Test Error
$\frac{100}{873}$	0.0126	0.0200	0.0138	0.0220
$\frac{500}{873}$	0.0310	0.0420	0.0252	0.0300
$\frac{700}{873}$	0.0252	0.0300	0.0161	0.0140

2.

a)

I get similar results in terms of training and test accuracies with the dual as with the prime however I get vastly different weights and bias terms. My weights in the primal were all in the thousands where as the dual gives single digit values. That being said comparing their values relative to each other and they're of a similar relative value. As far as what happens as C changes the dual implementation gets worse for increasing values of C but honestly not much worse.

b)

Included below is a table of my Gaussian Kernel results for varying C and γ values. The best combinations are with lower values of gamma with lower C values as well. Most of the gammas with C of $\frac{100}{873}$ or $\frac{500}{753}$ are under .2% error which is kind of amazing to me. Comparing to the linear SVM, most of these combinations out preform it by an order of magnitude, which again is kind of insane to consider but makes sense. The obvious conclusion is that mapping to higher dimensions and correctly selecting your hyperparameters gives amazing results, and that the hyperparameters can make a huge difference in those results.

C	γ	Train Error	Test Error
$\frac{100}{873}$	0.1	0.000	0.002
	0.5	0.000	0.002
	1.0	0.000	0.002
	5.0	0.008	0.006
	100.0	0.234	0.240
$\frac{500}{873}$	0.1	0.000	0.002
	0.5	0.000	0.002
	1.0	0.000	0.002
	5.0	0.032	0.036
	100.0	0.258	0.254
$\frac{700}{873}$	0.1	0.000	0.002
	0.5	0.003	0.008
	1.0	0.002	0.012
	5.0	0.034	0.038
	100.0	0.259	0.256

c)

The table below contains the support vector count as well as the overlap for all C and γ values. As we increase the gamma size it reduces the number of SVs required to separate the data, additionally increasing the gamma reduces the number of overlapping SVs. I'm assuming this is caused by increasing the size of the kernel and making it so fewer SVs can encapsulate the same amount of data.

C	Gamma	Number of Support Vectors	Number of Overlapping Support Vectors
$\frac{100}{873}$	0.1	867	NaN
	0.5	783	782.0
	1.0	746	746.0
	5.0	723	689.0
	100.0	636	573.0
$\frac{500}{873}$	0.1	862	NaN
	0.5	779	779.0
	1.0	717	652.0
	5.0	492	456.0
	100.0	589	393.0
$\frac{700}{873}$	0.1	860	NaN
	0.5	783	782.0
	1.0	612	551.0
	5.0	476	453.0
	100.0	588	381.0