CS771A: Machine Learning *Your Name & Roll no:*

# Mid-sem Exam (Sem-I, 2016)

## Instructions:

- This question paper is worth a total of 120 marks.
- Total time is 2 hrs + 30 mins
- Please write your name and roll number at the top of this sheet as well as on the provided answer copy.
- The True/False questions have to be answered on this question sheet itself. Please write T/F in the square bracket printed at the beginning of each question. This question sheet must also be submitted.
- To answer the other parts of the question paper, please use the provided answer copy.
- For rough work, please use pages at the back of answer copy. Extra sheets can be provided if needed.

# Questions begin from the next page

# 1 True/False Problems (20 questions, 20 marks)

Please read each statement below carefully and then mark T/F in the square bracket. There are no partial marks for this section, so please do not explain your answer.

For some questions, I have given a brief explanation for the T/F answer or have highlighted the important word(s) in the question text itself.

1. [**T**] The more you regularize a supervised learning model, the larger the training error becomes.
2. [**T**] For DT, time required to predict the label of a test example does not depend on the number of training examples. (It depends on the depth of the tree.)
3. [**F**] The Perceptron can be turned into a maximum margin classifier by modifying the Perceptron mistake condition from $y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) < 0$ to $y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) < \tau$ for some non-negative $\tau$.
4. [**F**] $K$-means, run multiple times with different initializations, will find the same solution every time.
5. [**F**] PCA reduces data dimensionality by choosing a subset of relevant features from the original data.
6. [**T**] If run on the same training dataset, a soft-margin SVM is expected to have a smaller generalization error than that of a hard-margin SVM.
7. [**T**] A kernel function defined as $k = 2k_1 - 3k_2$ is not guaranteed to be a valid kernel function even if $k_1$ and $k_2$ are valid kernel functions.
8. [**F**] An $\ell_2$ regularizer on a weight vector $\boldsymbol{w}$ encourages $\boldsymbol{w}$ to have very few nonzeros entries.
9. [**F**] $K$NN with Euclidean distance reduces to a linear classification model. (Even with a Euclidean distance, $K$NN can learn nonlinear decision boundaries; recall lecture-2 and the Voronoi diagrams !)
10. [**T**] The Perceptron is guaranteed to converge in a finite number of iterations for separable data.
11. [**T**] MAP estimation is, in general, less prone to overfitting than MLE.
12. [**F**] For a logistic regression model for binary classification, if the predicted probability $p(y = 1|\boldsymbol{x}, \boldsymbol{w})$ is close to zero, it means that the predicted label is ambiguous/uncertain. (close to zero probability for class 1 means close to one probability for class 0, which actually means that it's a confident prediction)
13. [**F**] Stochastic gradient descent runs faster than standard gradient descent because it updates the components of weight vector $\boldsymbol{w} \in \mathbb{R}^D$ as one component at a time. (No, because the stochastic gradients can be cheaply computed using only one or a small number of examples)
14. [**T**] For $K$NN regression/classification, increasing the value of $K$ does not lead to overfitting.
15. [**F**] The computational cost of computing inner products in a kernel induced $M$-dimensional space would scale as $\mathcal{O}(\sqrt{M})$. (Inner product in the $M$ dim. $\phi$ space can be directly computed using $k(\boldsymbol{x}, \boldsymbol{z})$)
16. [**F**] PCA guarantees perfect reconstruction of an input $\boldsymbol{x} \in \mathbb{R}^D$ from its $K < D$ dimensional projection.
17. [**T**] Increasing the slack penalty parameter $C$ in the soft-margin SVM will result in fewer support vectors in the final solution.
18. [**F**] Logistic regression, unlike linear regression, doesn't have a closed form solution for the weight vector because the logistic loss is non-convex. (It is due to purely algebraic reasons)
19. [**T**] Doing MLE for a probabilistic linear regression model with Gaussian likelihood will give the same solution as an unregularized least squares linear regression model.
20. [**F**] Kernel $K$-means has no additional computational overhead as compared to standard $K$-means because kernels can compute inner products efficiently. (Distance computations in kernel version are relatively more expensive, despite the fact that $k(\boldsymbol{x}_n, \boldsymbol{x}_m)$ can be efficiently computed; lec-10, slide 22)

## 2 Short Answer Problems (10 questions, 30 marks)

Note: For this part, 1-2 sentences (and/or maybe 1-2 equations) should suffice to answer each question. Therefore, be concise and precise in your answers, so that you don't waste time unnecessarily.

1. Why won't you use a regression model for predicting whether an email is spam or not?

   **Answer:** It's a classification problem since the output is discrete-valued (binary in this case). Regression models on the other hand assume real-valued outputs.

2. Suppose you have $N$ input examples and you have used an RBF kernel $k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \exp(-\gamma||\boldsymbol{x}_n - \boldsymbol{x}_m||^2)$ to form a kernel (similarity) matrix $\mathbf{K}$ of size $N \times N$. Suppose the kernel's hyperparameter $\gamma \to \infty$. What will be the trace of $\mathbf{K}$? What will be the rank of $\mathbf{K}$?

   **Answer:** Since $\gamma \to \infty$, we will have $\exp(-\gamma||\boldsymbol{x}_n - \boldsymbol{x}_m||^2) = 0$ for all entries of $\mathbf{K}$, except the diagonal entries where $\boldsymbol{x}_n = \boldsymbol{x}_m$, in which case $\exp(-\gamma||\boldsymbol{x}_n - \boldsymbol{x}_m||^2) = 1$. This makes $\mathbf{K}$ an identity matrix which has trace $= N$ and rank $= N$.

3. Suppose you know the importance of each of the $D$ features in your data and this is given to you as a vector of positive numbers $\gamma_1, \gamma_2, \ldots, \gamma_D$. Suggest a way to utilize this information in any learning algorithm that uses distance/similarities.

   **Answer:** We can use the given feature importances in our distance/similarity computations, e.g., Euclidean distance between two data points $\boldsymbol{x} \in \mathbb{R}^D$ and $\boldsymbol{z} \in \mathbb{R}^D$ can be modified as $\sum_{d=1}^{D} \gamma_d (x_d - z_d)^2$.

4. What is underfitting? When might it happen?

   **Answer:** Underfitting means that the model is too simple as compared to the complexity of data. This can also happen if you "over-regularize" a rich-enough model so much that it becomes too simple.

5. Why solving linear regression in the dual (where we estimate the dual variables $\alpha_n$'s) and writing $\boldsymbol{w}$ as $\boldsymbol{w} = \sum_{n=1}^{N} \alpha_n \boldsymbol{x}_n$ may be preferable, as opposed to solving $\boldsymbol{w} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_D)^{-1} \mathbf{X}^\top \boldsymbol{y}$?

   **Answer:** In the dual representation, to learn $\boldsymbol{w}$, you only need to learn the $\alpha_n$'s which requires an $N \times N$ matrix inversion. This is more efficient than doing $\boldsymbol{w} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_D)^{-1} \mathbf{X}^\top \boldsymbol{y}$, when $N \ll D$.

6. Sort the following classification algorithms by training time and then by test time (fastest first, slowest last): 1NN, perceptron, SVM with RBF kernel, decision tree with bounded depth of 5. (Assume that we have 10,000 training examples in 500 dimensions.)

   **Answer:** Training times: 1NN (no training needed), DT, Perceptron, SVM with RBF kernel. Test time: DT, Perceptron, SVM with RBF kernel, 1NN.

7. The standard perceptron update is $\boldsymbol{w} = \boldsymbol{w} + y_n \boldsymbol{x}_n$ when a mistake is made. Suppose we were to add a small learning rate $\eta > 0$ so that the update became $\boldsymbol{w} = \boldsymbol{w} + \eta y_n \boldsymbol{x}_n$. Does this change the final learned classifier? If so, how? If not, why not?

   **Answer:** It won't change the direction of the learned weight vector. The final weight vector will be scaled by a constant $\eta$.

8. Write down the expression for the stochastic gradient descent based update rule for the standard linear regression model which uses a squared loss (ignore the regularization term).

   **Answer:** For squared loss, the stochastic gradient (using a single example) is simply $-2(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)\boldsymbol{x}_n$. Thus the update will be $\boldsymbol{w} = \boldsymbol{w} + \eta(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)\boldsymbol{x}_n$ (the "2" is subsumed in $\eta$)

9. Why maximizing the margin in SVM would lead to small values of entries in the SVM weight vector?

   **Answer:** Margin is inversely proportional to the $\ell_2$ norm, so maximizing the margin is equivalent to minimizing the $\ell_2$ norm, which in turn encourages small values of the entries in the weight vector.

10. Why it may not always be a good idea to apply PCA on some data matrix $\mathbf{X}$ where the final goal could be to train a classification model?

    **Answer:** Because the directions of maximum variance may not necessarily be the directions along which the classes are separable (projection based solely on the variance criteria might mix the classes).

# 3 Medium Length Answer Problems (5 questions, 40 marks)

1. Consider a kernel $k(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{x}^\top \boldsymbol{z} + 2(\boldsymbol{x}^\top \boldsymbol{z})^2$. As we know, each kernel has an associated feature mapping $\boldsymbol{\phi}$ such that $k(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{z})$. Assume that each example has two features (so $\boldsymbol{x} = \{x_1, x_2\}, \boldsymbol{z} = \{z_1, z_2\}$). Write down the feature mapping $\boldsymbol{\phi}$ associated with the kernel defined above.

   **Answer:** We have $k(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{x}^\top \boldsymbol{z} + 2(\boldsymbol{x}^\top \boldsymbol{z})^2 = (x_1 z_1 + x_2 z_2) + 2(x_1 z_1 + x_2 z_2)^2 = x_1 z_1 + x_2 z_2 + 2x_1^2 z_1^2 + 2x_2^2 z_2^2 + 4x_1 z_1 x_2 z_2 = \phi(\boldsymbol{x})^\top \phi(\boldsymbol{z})$, where $\phi$ will have the form $\phi(\boldsymbol{x}) = [x_1 \; x_2 \; \sqrt{2}x_1^2 \; \sqrt{2}x_2^2 \; 2x_1 x_2]$

2. PCA finds projection directions along which the projected data has the largest variances. Suppose you have a supervised learning problem with feature matrix $\mathbf{X}$ of size $D \times N$ ($D$ features, $N$ examples) and binary label vector $\boldsymbol{y}$ of size $N \times 1$. Assume $\mathbf{X}$ is already centered. We want to reduce the data dimensionality using the usual PCA criterion but *also* want the projection directions to preserve class separation. Intuitively, for any two points $\boldsymbol{x}_n$ and $\boldsymbol{x}_m$, their projections $\boldsymbol{z}_n = \boldsymbol{u}^\top \boldsymbol{x}_n$ and $\boldsymbol{z}_m = \boldsymbol{u}^\top \boldsymbol{x}_m$ (a single projection direction $\boldsymbol{u} \in \mathbb{R}^D$ assumed for simplicity) should have high correlations with the corresponding labels $y_n$ and $y_m$ (or, equivalently, if two points have the same label, their projections should be close to each other). How would you modify PCA's "maximizing the variance" objective function to achieve this goal. **You are not required to solve the objective**.

   **Answer:** Recall that PCA maximizes the objective $\boldsymbol{u}^\top \mathbf{S} \boldsymbol{u}$, where $\mathbf{S} = \frac{1}{N}\mathbf{X}\mathbf{X}^\top$ is the data covariance matrix, w.r.t. $\boldsymbol{u}$ (subject to the unit norm constraint on the projection direction $\boldsymbol{u}$).

   To preserve the class separation, we would also like the projections $\boldsymbol{z} = [\boldsymbol{u}^\top \boldsymbol{x}_1, \ldots, \boldsymbol{u}^\top \boldsymbol{x}_N]^\top = \mathbf{X}^\top \boldsymbol{u}$ to be highly correlated with the outputs $\boldsymbol{y} = [y_1 \;\; y_2 \;\; \ldots y_N]^\top$. One way to express this correlation would be as $\boldsymbol{z}^\top \boldsymbol{y} = \boldsymbol{u}^\top \mathbf{X} \boldsymbol{y} = (\boldsymbol{u}^\top \mathbf{X} \boldsymbol{y})^2 = \boldsymbol{u}^\top \mathbf{X} \boldsymbol{y} \boldsymbol{y}^\top \mathbf{X}^\top \boldsymbol{u}$. We want this to have a large value.

   We can then define the final objective as something like a weighted sum of the original PCA objective and this criterion, e.g., $\beta \boldsymbol{u}^\top \mathbf{S} \boldsymbol{u} + (1-\beta) \boldsymbol{u}^\top \mathbf{X} \boldsymbol{y} \boldsymbol{y}^\top \mathbf{X}^\top \boldsymbol{u}$, where $\beta$ is a trade-off hyperparameter controlling the variance vs class separation objectives. Of course, this is just one of the many way of doing this (other reasonable solutions will also be acceptable; it's basically "supervised" PCA).

3. Suppose you train a linear regression model with $\ell_2$ regularizer and learn a weight vector $\boldsymbol{w}$. Now suppose that you change all the inputs by *duplicating* the first feature. That is, if you originally had 100 features, you now have 101 features where feature 1 and feature 2 are *identical*. Now, you retrain the model and get a weight vector $\tilde{\boldsymbol{w}}$. Suppose the new $\tilde{\boldsymbol{w}}$ gives the *same training error* as the earlier $\boldsymbol{w}$. How do $\boldsymbol{w}$ and $\tilde{\boldsymbol{w}}$ relate with each other? In particular, how does the first component $w_1$ of $\boldsymbol{w}$ relate to *each* of the first two components $\tilde{w}_1$ and $\tilde{w}_2$ of $\tilde{\boldsymbol{w}}$? (**Hint:** Note that we have an $\ell_2$ regularizer).

   **Answer:** Original feature vector $\boldsymbol{x} = [x_1 \; x_2 \ldots \; x_{100}]$. New feature vector with first feature duplicated will be $\tilde{\boldsymbol{x}} = [x_1 \; \boldsymbol{x}]$. Since the model is linear and the training (i.e., empirical) loss is the same, we must have that $\boldsymbol{w}^\top \boldsymbol{x} = \tilde{\boldsymbol{w}}^\top \tilde{\boldsymbol{x}}$, which expands to $w_1 x_1 + \sum_{d=2}^{100} w_d x_d = \tilde{w}_1 x_1 + \tilde{w}_2 x_1 + \sum_{d=2}^{100} \tilde{w}_{d+1} x_d$. This implies that $w_1 = \tilde{w}_1 + \tilde{w}_2$. Since we have an $\ell_2$ regularizer on $\tilde{\boldsymbol{w}}$, $\tilde{w}_1^2 + \tilde{w}_2^2$ would be minimized as well. Since $\tilde{w}_1 + \tilde{w}_2 = w_1$ is fixed, the sum of squares $\tilde{w}_1^2 + \tilde{w}_2^2$, will be minimized when both components are equal, i.e., $\tilde{w}_1 = \tilde{w}_2 = w_1/2$.

4. Due to the form of Perceptron updates $\boldsymbol{w} = \boldsymbol{w} + y_n \boldsymbol{x}_n$ (ignore the bias $b$), the weight vector learned by Perceptron can be written as $\boldsymbol{w} = \sum_{n=1}^N \alpha_n y_n \boldsymbol{x}_n$, where $\alpha_n$ is the number of times Perceptron makes a mistake on example $n$. Suppose our goal is to make Perceptron learn nonlinear boundaries, using a kernel $k$ with feature map $\phi$. Modify the standard Perceptron algorithm to do this. In particular, for this kernelized variant of the Perceptron algorithm (1) Give the initialization, (2) Give the mistake condition, and (3) Give the update equation. **Hint:** Use the fact that doing mistake-driven updates for $\boldsymbol{w}$ is equivalent to doing the same for the $\alpha_n$'s.

   **Answer:** We can represented $\boldsymbol{w}$ is using the $N$ dimensional vector $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_N]$.

   (1) Initialization: You can initialize $\boldsymbol{\alpha}$ as a vector of all zeros.

   (2) Mistake Condition: Using $\boldsymbol{w} = \sum_{m=1}^N \alpha_m y_m \boldsymbol{x}_m$, the Perceptron mistake condition $y_n \boldsymbol{w}^\top \boldsymbol{x}_n < 0$ can be re-written as $y_n \sum_{m=1}^N \alpha_m y_m \boldsymbol{x}_m^\top \boldsymbol{x}_n < 0$. In the kernelized version, $\boldsymbol{x}_m^\top \boldsymbol{x}_n$ is replaced by $k(\boldsymbol{x}_m, \boldsymbol{x}_n)$. The mistake condition thus becomes $y_n \sum_{m=1}^N \alpha_m y_m k(\boldsymbol{x}_m, \boldsymbol{x}_n) < 0$

   (3) Update Equation: If mistake made on example $(\boldsymbol{x}_n, y_n)$, the update $\alpha_n$ as $\alpha_n = \alpha_n + 1$.

5. The Poisson distribution is a distribution over positive count values. It has the form $p(k|\lambda) = \frac{\lambda^k e^{-\lambda}}{k!}$, where $k$ is the count and $\lambda$ is the (single) parameter of the Poisson. Suppose we have a bunch of count data (for instance, the number of cars to pass an intersection on a given day, measured on $N$ many days) denoted by $k_1, k_2, \ldots, k_N$ (each a positive integer). Compute the MLE for $\lambda$ given this data.

   **Answer:** MLE will find $\lambda$ that maximizes the sum of log likelihoods over the $N$ observations

   $$\text{Sum of log likelihoods:} \quad L(\lambda) = \sum_{n=1}^{N} \log \frac{\lambda^{k_n} e^{-\lambda}}{k_n!} = \sum_{n=1}^{N} [k_n \log \lambda - \lambda] \quad (-\log k_n! \text{ can be ignored})$$

   The MLE will be given by $\hat{\lambda} = \arg\max_\lambda L(\lambda)$. Taking the derivative w.r.t. $\lambda$ we have $\sum_{n=1}^{N} [\frac{k_n}{\lambda} - 1] = 0$. Setting it to zero gives $\hat{\lambda} = \frac{1}{N} \sum_{n=1}^{N} k_n$

# 4 Long Answer Problems (2 questions, 30 marks)

1. Consider a slight variation of the standard probabilistic linear regression model where the likelihood function is given by $p(y_n|\boldsymbol{w}, \boldsymbol{x}_n) = \frac{1}{\sqrt{2\pi\sigma_n^2}} \exp(-\frac{(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2}{2\sigma_n^2})$. Each input $\boldsymbol{x}_n$ is $D$ dimensional and each output $y_n$ is a scalar. Also assume the prior distribution over the weight vector $\boldsymbol{w} \in \mathbb{R}^D$ to be the following $D$ dimensional multivariate Gaussian $p(\boldsymbol{w}) = \frac{1}{\sqrt{(2\pi)^D |\boldsymbol{\Lambda}|}} \exp(-\frac{1}{2} \boldsymbol{w}^\top \boldsymbol{\Lambda}^{-1} \boldsymbol{w})$, where $\boldsymbol{\Lambda}$ is a $D \times D$ diagonal matrix with the diagonal entries being $\lambda_1, \lambda_2, \ldots, \lambda_D$. In this model, only $\boldsymbol{w}$ is unknown and the other hyperparameters (the $\sigma_n$'s and $\lambda_d$'s) are assumed known.

   For this model, write down the MAP objective function and simplify it as much as possible. **You do not need to solve the objective**. Give an interpretation of the final objective function. In particular, what roles the hyperparameters $\sigma_n$'s and $\lambda_d$'s are playing here?

   Note: If it helps your calculations, note that the multivariate Gaussian prior $\frac{1}{\sqrt{(2\pi)^D |\boldsymbol{\Lambda}|}} \exp(-\frac{1}{2} \boldsymbol{w}^\top \boldsymbol{\Lambda}^{-1} \boldsymbol{w})$ with diagonal covariance $\Lambda$ can be written as $\prod_{d=1}^{D} \frac{1}{\sqrt{2\pi\lambda_d}} \exp(-\frac{w_d^2}{2\lambda_d})$

   **Answer:** Recall that the MAP problem basically maximizes "log likelihood + log prior" or minimizes "negative log likelihood - log prior". The negative log likelihood for *each* data point is $\frac{(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2}{2\sigma_n^2}$ (ignoring terms that are constant w.r.t. $\boldsymbol{w}$), and the negative log prior is $-\log \prod_{d=1}^{D} \frac{1}{\sqrt{2\pi\lambda_d}} \exp(-\frac{w_d^2}{2\lambda_d})$, which, ignoring the terms that are constant w.r.t. $\boldsymbol{w}$, is simply $\sum_{d=1}^{D} \frac{w_d^2}{2\lambda_d}$. Assuming there are a total of $N$ observations, the MAP problem reduces to minimizing

   $$L(\boldsymbol{w}) = \sum_{n=1}^{N} \frac{(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2}{2\sigma_n^2} + \sum_{d=1}^{D} \frac{w_d^2}{2\lambda_d}$$

   This now has a simple interpretation. The empirical loss part above is like a "weighted" least squares objective (recall one of the HW1 problems) where error on each example $(\boldsymbol{x}_n, y_n)$ is given a different weight $1/2\sigma_n^2$ (note that it will also prevent the outlier/noisy examples, which will have large noise variance $\sigma_n^2$, from influencing the solution too much). The regularizer is also a variation of the standard $\ell_2$ regularizer - here each entry $w_d$ of $\boldsymbol{w}$ is penalized differently based on the corresponding $\lambda_d$ value.

2. Sometimes it costs us a lot more to classify negative points as positive than positive points as negative. (Think: we are predicting if someone has cancer. We would rather err on the side of caution (predicting "yes" when the answer is "no") than vice versa.) One way of expressing this in a support vector machine is to give different costs to the two kinds of mis-classification. The primal formulation of this is:

   $$\min_{\boldsymbol{w}, b, \boldsymbol{\xi}} \quad \frac{||\boldsymbol{w}||^2}{2} + \sum_{n=1}^{N} C_{y_n} \xi_n$$

subject to $y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \geq 1 - \xi_n$ and $\xi_n \geq 0$, $\forall n$.

The only difference is that instead of one cost parameter $C$, there are two, $C_{+1}$ and $C_{-1}$, representing the costs of misclassifying positive examples and misclassifying negative examples, respectively.

Write down the Lagrangian problem of this modified SVM. Take derivatives w.r.t. the primal variables and construct the dual, namely, the maximization problem that depends only on the dual variables $\boldsymbol{\alpha}$, rather than the primal variables. Explain (intuitively) how this differs from the standard SVM dual problem; in particular, how the $C$ variables differ between the two duals.

**Answer:** The Lagrangian will be

$$\mathcal{L}(\boldsymbol{w}, b, \xi, \alpha, \beta) = \frac{\boldsymbol{w}^\top \boldsymbol{w}}{2} + \sum_{n=1}^{N} C_{y_n}\xi_n + \sum_{n=1}^{N} \alpha_n\{1 - y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) - \xi_n\} - \sum_{n=1}^{N} \beta_n\xi_n$$

Taking derivatives w.r.t. the primal variables $\boldsymbol{w}$, $b$, $\xi_n$, and setting to zero, we get

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = 0 \Rightarrow \boldsymbol{w} = \sum_{n=1}^{N} \alpha_n y_n \boldsymbol{x}_n, \quad \frac{\partial \mathcal{L}}{\partial b} = 0 \Rightarrow \sum_{n=1}^{N} \alpha_n y_n = 0, \quad \frac{\partial \mathcal{L}}{\partial \xi_n} = 0 \Rightarrow C_{y_n} - \alpha_n - \beta_n = 0$$

From the third condition, using the fact that $\beta_n \geq 0$, we have $\alpha_n \leq C_{y_n}$

Also plugging in the expressions of $\boldsymbol{w}$ and $b$ in the Lagrangian, and using $\sum_{n=1}^{N} \alpha_n y_n = 0$, we have

$$\frac{1}{2}(\sum_{n=1}^{N} \alpha_n y_n \boldsymbol{x}_n)^\top (\sum_{m=1}^{N} \alpha_m y_m \boldsymbol{x}_m) + \sum_{n=1}^{N} \alpha_n - \sum_{n=1}^{N} \alpha_n y_n (\sum_{m=1}^{N} \alpha_m y_m \boldsymbol{x}_m)^\top \boldsymbol{x}_n + \sum_{n=1}^{N} (C_{y_n} - \beta_n)\xi_n - \sum_{n=1}^{N} \alpha_n \xi_n$$

The last two terms will vanish since $C_{y_n} - \beta_n = \alpha_n$, and we will be left with the following dual objective

$$\mathcal{L}(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{M} \alpha_n \alpha_m y_n y_m \boldsymbol{x}_n^\top \boldsymbol{x}_m \qquad \text{s.t. } \sum_{n=1}^{N} \alpha_n y_n = 0 \quad \text{and } \alpha_n \leq C_{y_n}, \beta_n \geq 0 \ \forall n$$

We finally solve $\max_{\boldsymbol{\alpha},\boldsymbol{\beta}} \mathcal{L}(\boldsymbol{\alpha}, \boldsymbol{\beta})$, subject to constraints $\sum_{n=1}^{N} \alpha_n y_n = 0, \alpha_n \leq C_{y_n}, \beta_n \geq 0 \ \forall n$

The only difference from the standard SVM dual problem (which has the same $C$ for every example and $\alpha_n \leq C$) is that now we have different caps on the $\alpha_n$'s of positive and negative training examples. The $\alpha_n$'s of positive examples can't exceed $C_{+1}$ and $\alpha_n$'s of negative examples can't exceed $C_{-1}$. Since $\alpha_n$ is sort of the "importance" of example $n$ in the SVM solution (recall that $\boldsymbol{w} = \sum_{n=1}^{N} \alpha_n y_n \boldsymbol{x}_n$), it means that you are giving training examples from positive and negative classes *unequal* importances (depending on the relative values of $C_{+1}$ or $C_{-1}$).