CPSC 340 Final (Fall 2015)

Name:

Student Number:

Please enter your information above, turn off cellphones, space yourselves out throughout the room, and wait until the official start of the exam to begin. You can raise your hand to ask a question, and please look up occasionally in case there are clarifications written on the projector (the time remaining will also be written on the projector). You are welcome to (quietly) leave early if you finish early.

The final consists of 9 questions, and they will all be equally weighted in the marking scheme. Note that some question have multiple parts, written as **(a)**, **(b)**, **(c)** and **(d)**. *Clearly mark* where you are answering each part, *show your work/reasoning for each part*, and make sure to check that you answered all parts before handing in your final.

Good luck!

1 Training/Validation/Testing

You are asked by a client to build a system that solves a binary classification problem. They give you 3000 training examples and a set of 100 features for each example. You have been assured that the examples have been generated in way that makes them IID, and the examples have been given to you *sorted* based on the values of the first feature. The client not only wants an accurate model, but they want an estimate of the accuracy of the final model.

Assume that the data is stored in 3000 by 100 matrix X, and the labels are stored in a 3000 by 1 vector y. As in the assignments, assume that you have a 'model' function that depends on a parameter 'k' with the following interface:

- model = train(X,y,k); % Train model on $\{X, y\}$ with parameter k
- yhat = predict(model,Xhat); % Predicts using the model on $Xhat$.

Assume that k can take any integer value from 1 to 10.

Give pseudo-code for a training/validation/testing procedure that:

(a) Chooses a good value of 'k'.

(b) Reports an unbiased estimate of the accuracy of the final model.

Answer:

Since the examples were sorted, you first need to randomize the order of the examples. One way to do this is to place the rows of X in a random order, re-ordering y using the same order.

Next split the examples into a training, validation, and testing set. For example, you could use the first 1000 random examples as the training set (Xtrain, ytrain), the next 1000 fas the validation set (Xvalidate, yvalidate), and the final 1000 as the testing set (Xtest, ytest). If you did not place the examples in a random order, you could choose these three sets randomly from the examples, but *ensuring that there is no overlap* between the three sets.

For each value of k from 1 to 10, perform the following:

- model = train(Xtrain,ytrain,k).
- yhat = predict(model,Xvalidate).
- err(k)= sum of elements where yhat does not equal yvalidate.

Set k to be the minimum of err(k) across k.
(It's ok if they used cross-validation instead of a training/validation set.)

To report the estimated accuracy of the final model:

- yhat = predict(model,Xtest).
- accuracy = sum of elements where yhat equals ytest, divided by length of ytest.

2 KNN, Naive Bayes, and Softmax

Consider the dataset below, which has 10 training examples and 2 features:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 2 \\ 2 \\ 2 \\ 3 \\ 3 \\ 3 \\ 3 \end{bmatrix}.$$

Suppose that you want to classify the following test example:

$$\hat{x} = \begin{bmatrix} 1 & 1 \end{bmatrix}.$$

(a) What class label would we assign to the test example if we used a k-nearest neighours classifier, with $k = 3$ and the Euclidean distance measure?

(b) What class label would we assign to the test example under a naive Bayes model? (Show your work.)

(c) Suppose we fit a multi-class linear classifier using the softmax loss, and we obtain the following weight matrix:

$$W = \begin{bmatrix} +2 & +2 & +3 \\ -1 & +2 & -1 \end{bmatrix}$$

Under this model, what class label would we assign to the test example? (Show your work.)

Answer:
(a) There are three training examples that have a distance of zero to the test example. Two out of three of them have the label '2', so we would classify the example as a '2'.

(b) By counting we have

$$\begin{aligned}
p(y=1) &= 3/10 \\
p(y=2) &= 3/10 \\
p(y=3) &= 4/10 \\
p(x_1=1|y=1) &= 2/3 \\
p(x_2=1|y=1) &= 1/3 \\
p(x_1=1|y=2) &= 2/3 \\
p(x_2=1|y=2) &= 2/3 \\
p(x_1=1|y=3) &= 1 \\
p(x_2=1|y=3) &= 1/4.
\end{aligned}$$

Putting these together we get

$$\begin{aligned}
p(y=1|x_1=1,x_2=1) &\propto p(y=1)p(x_1=1|y=1)p(x_2=1|y=1) = (3/10)(2/3)(1/3) = 0.0\bar{6} \\
p(y=2|x_1=1,x_2=1) &\propto p(y=2)p(x_1=1|y=2)p(x_2=1|y=2) = (3/10)(2/3)(2/3) = 0.1\bar{3} \\
p(y=3|x_1=1,x_2=1) &\propto p(y=3)p(x_1=1|y=3)p(x_2=1|y=3) = (4/10)(1)(1/4) = 0.10
\end{aligned}$$

So we would classify the example as a '2'.

(c) This model bases its decision of maximizing the inner-product, $w_c^T\hat{x}$.

For class 1 we have

$$w_1^T\hat{x} = (+2)1 + (-1)1 = 1.$$

For class 2 we have

$$w_2^T\hat{x} = (+2)1 + (+2)1 = 4.$$

For class 3 we have

$$w_3^T\hat{x} = (+3)1 + (-1)1 = 2.$$

So this model would also predict '2'.

3 Parametric vs. Non-Parametric

Make a table with a column labeled 'parametric' and a column labeled 'non-parametric'. Place each of the following methods into one of the columns:

- Mean of a list of numbers.
- Scatterplot.
- Depth-10 decision trees.
- 3-nearest neighbours.
- Naive Bayes with 1000 variables.
- Random forests with 10 random trees of depth 10.
- K-means with 5 means.
- Density-based clustering.
- Linear regression with linear basis.
- Linear regression with RBF basis.
- Principal component analysis.
- Non-negative matrix factorization.
- Sammon mapping.
- Neural networks.

Parametric	Non-parametric
Mean of a list of numbers	Scatterplot
Depth-10 decision trees	3-nearest neighbours
Naive Bayes with 1000 variables	
Random forests with 10 random trees of depth 10	
K-means with 5 means	Density-based clustering
Linear regression with linear basis	Linear regression with RBF basis
Principal component analysis	
Non-negative matrix factorization	Sammon mapping
Neural network	

4 L1-Regularized Latent-Factor Model

We have a matrix X, where we have observed a subset of its individual elements. Let $\mathcal{R}$ be the set of indices (i, j) where we have observed the element x_{ij}. We want to build a model that predicts the missing entries, so we use a latent-factor model with an L1-regularizer on the coefficients W and a separate L2-regularizer on the coefficients Z,

$$\underset{W \in \mathbb{R}^{k \times d}, Z \in \mathbb{R}^{n \times k}}{\operatorname{argmin}} \sum_{(i,j) \in \mathcal{R}} \left[\frac{1}{2}(x_{ij} - w_j^T z_i)^2 \right] + \lambda_W \sum_{j=1}^{d} \left[\|w_j\|_1 \right] + \lambda_Z \sum_{i=1}^{n} \left[\|z_i\|^2 \right],$$

where the regularization parameters satisfy $\lambda_W > 0$ and $\lambda_Z > 0$.

(a) What is the affect of λ_W on the sparsity of the parameters W and Z? What is the effect of λ_Z on the sparsity of W and Z?

(b) What is the affect of λ_Z on the two parts of the fundamental trade-off in machine learning? What is the effect of k on the two parts?

(c) Would the answers to (b) change if $\lambda_W = 0$?

(d) Suppose each element of the matrix X is either $+1$ or -1 and our goal is to build a model that makes the sign of $w_j^T z_i$ match the sign of x_{ij}. Write down a (continuous) objective function that would be more suitable.

Answer:
(a)

1. As λ_W increases, W will become more sparse.
2. The sparsity of the matrix Z is not affected by λ_W.
3. The sparsity of W and Z are not affected by λ_Z.

(b)

1. As λ_W *increases*, there is a heavier bias towards zero and the *training error will go up*. However, the *training error will become a better approximation of the test error*
2. As k *increases*, the model becomes more complicated so the *training error will go down*. However, the *training error will become a worse approximation of the test error*.

(c)

1. The effect of k would stay the same, since increasing it still allows more complicated models.
2. If $\lambda_Z = 0$, then we have no constraints on Z. This means we could cancel out any decrease in W by increasing Z. Thus, λ_W *no longer directly affects* the fundamental trade-off.

(d)

In this setting, it would make more sense to do something like the logistic loss,

$$\underset{W\in\mathbb{R}^{k\times d}, Z\in\mathbb{R}^{n\times k}}{\operatorname{argmin}} \sum_{(i,j)\in\mathcal{R}} \left[\log(1+\exp(-x_{ij}w_j^T z_i))\right] + \lambda_W \sum_{j=1}^{k} \left[\|w_j\|_1\right] + \lambda_Z \sum_{i=1}^{n} \left[\|z_i\|^2\right],$$

but the regularizers can stay the same.

5 Label Propagation

Consider a transductive learning setting where we have a set of labelled examples $y_i \in \{-1, +1\}$ for i ranging from 1 to n. We also have a set of t unlabeled examples that we would like to label. While we do not have features for any examples, we are given weights w_{ij} indicating how strongly we prefer unlabeled example i to have the same label as labeled example j, and another set of weights v_{ij} indicating how strongly we prefer unlabeled example i to have the same label as unlabeled example j. We'll assume that $v_{ij} = v_{ji}$ and $v_{ii} = 0$.

To find the labels of the unlabeled examples, a standard label propagation objective is

$$\underset{\hat{y}_1 \in \mathbb{R}, \hat{y}_2 \in \mathbb{R}, \dots, \hat{y}_t \in R}{\operatorname{argmin}} \frac{1}{2} \sum_{j=1}^{n} \sum_{i=1}^{t} \left[w_{ij}(y_j - \hat{y}_i)^2 \right] + \frac{1}{2} \sum_{i=1}^{t} \sum_{j=i+1}^{t} \left[v_{ij}(\hat{y}_j - \hat{y}_i)^2 \right] + \frac{\lambda}{2} \sum_{i=1}^{t} \hat{y}_i^2.$$

The regularization term encourages the predictions to be close to 0, so that the model becomes less confident in the labels of examples where we have to propagate labels quite far to reach them. Although we can fit this model with gradient descent, a standard approach to fitting it is by cycling through each of the $\hat{y}_i$ and updating them to their optimal value given the values of the remaining $\hat{y}_j$ for $j \neq i$.

(a) Derive the partial derivative of this objective function with respect to a particular $\hat{y}_i$.

(b) Derive the optimal value of a particular $\hat{y}_i$, given the values of the remaining $\hat{y}_j$ for $j \neq i$.

(c) Describe a procedure for selecting a good value of λ.

Answer:

(a) We have that

$$\frac{\partial}{\partial \hat{y}_i} = \sum_{j=1}^{n}[-w_{ij}(y_j - \hat{y}_i)] + \sum_{j=1}^{t}[-v_{ij}(\hat{y}_j - \hat{y}_i)] + \lambda \hat{y}_i,$$

where we've used that $(\hat{y}_i - \hat{y}_i) = 0$ and the symmetry $v_{ij} = v_{ji}$.

(b) We notice that this is a leasts squares problem in disguise. Equating the partial derivative to zero and moving terms not depending on $\hat{y}_i$ to one side we have

$$\sum_{j=1}^{n}[w_{ij}\hat{y}_i] + \sum_{j=1}^{t}[v_{ij}\hat{y}_i] + \lambda \hat{y}_i = \sum_{j=1}^{n}[w_{ij}y_j] + \sum_{j=1}^{t}[v_{ij}\hat{y}_j].$$

Taking $\hat{y}_i$ outside the sums on the right and then solving for it gives

$$\hat{y}_i = \frac{\sum_{j=1}^{n}[w_{ij}y_j] + \sum_{j=1}^{t}[v_{ij}\hat{y}_j]}{\sum_{j=1}^{n}[w_{ij}] + \sum_{j=1}^{t}[v_{ij}] + \lambda}$$

(Basically, we take a weighted combination of our neighbours and normalize by the weights plus some extra amount λ that moves $\hat{y}_i$ closer to zero.)

(c) To select λ, you could use part of the *labeled data* as a validation set or you could do cross-validation with the labeled set. You would then do label propagation with this bigger set of unlabeled examples and choose the λ that best predicts the labeled examples that were moved to the unlabeled set.

6 Outlierness Ratio

In class we defined an 'outlierness' ratio of an example $x_i \in \mathbb{R}^d$ for $i = 1$ to n. This ratio depends on the k-nearest neighbours, $N_k(x_i)$, and the average distance to these k-nearest neighbours

$$D_k(x_i) = \frac{1}{k} \sum_{j \in N_k(x_i)} \|x_i - x_j\|.$$

Given these definitions, the 'outlierness' ratio is defined by the quantity

$$O(x_i) = \frac{D_k(x_i)}{\frac{1}{k}\sum_{j \in N_k(x_i)} D_k(x_j)},$$

which roughly measures whether x_i is further away from its neighbours than its neighbours are from their neighbours (we'll assume that no points have the exact same distance from each other).

(a) If we want to compute this measure for a single example x_i, what is the cost of computing this measure in $O()$ notation in terms of n, d, and k (give your reasoning)?

(b) Consider the case where you don't have explicit x_i values, but you instead have an *undirected graph* defined on the examples, and each edge in the graph has a *similarity score* in the range $(0, 1]$ between examples. Describe how you could define something like the outlierness ratio in this setting. (You can assume that the graph is connected, but you should not assume that each point has at least k neighbours in the graph.)

Answer:

(a) Computing the distance between two points costs $O(d)$. Computing the distance of x_i to each of the n points thus costs $O(nd)$. Given these distances, we can find the k-nearest neighbours in $O(n)$ using the select algorithm and we can compute $D_k(x_i)$ given their distances in $O(k)$. Since $k \leq n$, we so far have $O(nd)$.

We next need to find the nearest neighbours and compute $D_k(x_j)$ for each of the k neighbours. Following the reasoning above, this costs $O(ndk)$. Given these values we can compute $O(x_i)$ in $O(k)$ so the total cost is $O(ndk)$.

(b) There are many possible answers here, but they need to address two issues.

The first is how to use a similarity score rather than a distance score. There are many possible ways to address this, but an obvious one is to take the inverse of the similarity score (highly-similar objects have a distance of 1, dissimilar objects get larger distances). Another obvious one would be to change the definition of the outlierness socre to directly work with similiarites, maybe doing something like

$$O(i) = \frac{\frac{1}{k}\sum_{j \in N_k(x_j)} S_k(x_j)}{S_k(i)},$$

where $S_k(i)$ is the average similarity of i to its k-nearest neighbours.

The second issue is how to define the k-nearest neighbours, and the similarity/distance of objects not directly connected in the graph.

If you convert to distances, an obvious way is to approximate the geodesic distance is by finding the shortest paths in the graph. You could then use the k-nearest neighbours according to the geodesic distance to compute the outlierness ratio.

If you modify the outlierness score to use similarities, then a logical approach might be to find the shortest path where you multiply the similarities along the path (rather than adding distances). You could then use the $k-$nearest neighbours according to this similarity to compute the outlierness ratio.

7 Principal Component Analysis

Consider the following dataset, containing 5 examples with 2 features each:

x_1	x_2
-2	-1
-1	0
0	1
1	2
2	3

(a) What is the first principal component?

(b) What is the (L2-norm) reconstruction error of the point (3,3)? (Show your work.)

(c) What is the (L2-norm) reconstruction error of the point (3,4)? (Show your work.)

Answer:

(a) We do not need to center the first variable but we need to center the second one. The mean of the second variable is 1 so the centered data looks like this:

x_1	x_2
-2	-2
-1	-1
0	0
1	1
2	2

We see that all the centered variables lie along the $x_2 = x_1$. The direction of this line is $(1, 1)$, but since we normalize the principal components to have a distance of one the first PC is $W_1 = (1/\sqrt{(2)}, (1/\sqrt{(2)}))$ so that $\sqrt{W_1} = \sqrt{(1/\sqrt{2})^2 + (1/\sqrt{2})^2} = \sqrt{1/2 + 1/2} = 1$.

(b) To get the low-dimensional representation, we first subtract the means and then multiply by W_c,

$$z = (3 - 0)/\sqrt{2} + (3 - 1)/\sqrt{2} = 5/\sqrt{2}.$$

To go back to the original space, we multiply this by W_c and add back the means:

$$\hat{x} = \frac{5}{\sqrt{2}}(1/\sqrt{2}, 1/\sqrt{2}) + (0, 1) = (5/2, 7/2) = (2.5, 3.5),$$

so the reconstruction error is

$$\sqrt{(2.5 - 3)^2 + (3 - 3.5)^2} = \sqrt{1/4 + 1/4} = 1/\sqrt{2}.$$

(c)

$$z = (3 - 0)/\sqrt{2} + (4 - 1)/\sqrt{2} = 6/\sqrt{2}.$$

$$\hat{x} = \frac{6}{\sqrt{2}}(1/\sqrt{2}, 1/\sqrt{2}) + (0, 1) = (6/2, 8/2) = (3, 4),$$

which is the same as the original point so the reconstruction error is 0.

8 Poisson Regression

Suppose we have a set of training examples (x_i, y_i) and we want to fit a linear model of the form $y_i \approx w^T x_i$. However, we do not want to use the squared error since the values in y_i represents *counts* (like 'number of Facebook likes'). So instead we assume that y_i follows a Poisson distribution with a mean of $\exp(w^T x_i)$,

$$p(y_i | w^T x_i) = \frac{\exp(y_i w^T x_i)\exp(-\exp(w^T x_i))}{y_i!}.$$

We want to find the w that maximizes these probabilities assuming that our examples are IID,

$$\underset{w \in \mathbb{R}^d}{\text{argmax}} \prod_{i=1}^n p(y_i | w^T x_i).$$

(a) Show how finding w corresponds to minimizing an additive loss function,

$$\underset{w \in \mathbb{R}^d}{\textbf{argmin}} \sum_{i=1}^n f(y_i, w^T x_i),$$

and derive the form of this loss function (simplifying as much as possible).

(b) If the largest value of y_i in the training set is k, what is the cost of evaluating this objective function in terms of n, d, and k?

(c) Given the parameters w and the features $\hat{x}$ for a new example, derive an efficient algorithm for finding a value of c that maximizes $p(\hat{y} = c | w^T \hat{x})$.

(Hint: try to discover the relationship between $p(\hat{y} = c | w^T \hat{x})$ and $p(\hat{y} = c - 1 | w^T \hat{x})$.)

Answer:

(a) We can transform the product into a sum by taking the log, giving

$$\operatorname*{argmax}_{w\in\mathbb{R}^d} \sum_{i=1}^{n} \log p(y_i|w^T x_i, b),$$

and by taking the negative we get a minimization problem

$$\operatorname*{argmin}_{w\in\mathbb{R}^d} -\sum_{i=1}^{n} \log p(y_i|w^T x_i, b),$$

Plugging in the definition of p we get

$$\operatorname*{argmin}_{w\in\mathbb{R}^d} -\sum_{i=1}^{n} y_i w^T x_i - \exp(w^T x_i) - \log(y_i!).$$

Notice that the last term does not depend on w so we can ignore it, giving us

$$\operatorname*{argmin}_{w\in\mathbb{R}^d} \sum_{i=1}^{n} (-y_i w^T x_i + \exp(w^T x_i)).$$

(b) The dominant cost is the n inner products of size d, so the cost is $O(nd)$, there is no dependence on k.

(c) First compute $\lambda = w^T x_i$, which costs $O(d)$ and is the bottleneck. We have that

$$\begin{aligned} p(y_i = c|\lambda) &= \frac{\exp(c\lambda)\exp(-\exp(\lambda))}{c!} \\ &= \frac{\exp((c-1)\lambda + \lambda)\exp(-\exp(\lambda))}{(c-1)!c} \\ &= p(y_i = c-1|\lambda)\frac{\exp(\lambda)}{c}. \end{aligned}$$

If $\exp(\lambda) > c$ then $p(y_i = c|\lambda) > p(y_i = c-1|\lambda)$ and if $\exp(\lambda) < c$ then $p(y_i = c|\lambda) < p(y_i = c-1|\lambda)$, so we can to take the maximum among the two integers that surround $\exp(\lambda)$, which can be found in $O(1)$. This gives a total cost of $O(d)$.

9 Stochastic Gradient

Using the L2-regularized logistic loss to fit a binary classifier corresponds to solving the optimization problem

$$\operatorname*{argmin}_{w\in\mathbb{R}^d} \sum_{i=1}^{n} [\log(1+\exp(-y_i w^T x_i))] + \frac{\lambda}{2} w^T w.$$

Using $f(w)$ to denote the objective function, the gradient of the objective function can be written in the form

$$\nabla f(w) = \sum_{i=1}^{n} [g(x_i, w) x_i + \frac{\lambda}{n} w].$$

for a function g that returns a scalar given the training example x_i and parameter vector w. The cost of computing g is $O(m)$ if x_i has m non-zero values, since it requires multiplying each non-zero element of x_i by the corresponding element of w, so in the worst case computing g costs $O(d)$.

(a) Write pseudo-code doing an iteration of stochastic gradient on this model with a constant step-size α. What is the cost of performing an iteration of stochastic gradient in terms of n and d? (You can assume that generating a random number between 1 and n costs $O(1)$.)

(b) How does the cost per iteration in part (a) change if each x_i has at most m non-zeroes?

(c) Show how we can reduce the cost in part (b) by representing w as the product of a scalar β and a vector v, so that $w = \beta v$.

Answer:

(a)

First, generate a random integer i between 1 and n.

Next, compute $g(x_i, w^t)$ for the random example i.

Update the parameters based on the gradient of that particular example,

$$w^{t+1} = w^t - \alpha(g(x_i, w^t)x_i + \frac{\lambda}{n}w).$$

The first step costs $O(1)$ by the assumption in the question. The second step costs $O(d)$ in the worst case, and updating the parameter costs $O(d)$ too. This gives a total cost of $O(d)$.

(b)

Stochastic gradient: The cost of computing $g(x_i, w^t)$ is now $O(m)$. However, since w will in general be dense the cost of the update remains $O(d)$. Thus, sparsity does not help and the cost remains $O(d)$.

(c)

Let's re-write the update in the form

$$w^{t+1} = \left(1 - \frac{\alpha\lambda}{n}\right) w^t - \alpha g(x_i, w^t)x_i,$$

or written in two steps as

$$w^{t+1/2} = \left(1 - \frac{\alpha\lambda}{n}\right) w^t,$$

$$w^{t+1} = w^{t+1/2} - \alpha g(x_i, w^t)x_i.$$

Now consider using the representation $\beta^t v^t = w^t$. To update to $w^{t+1/2}$ we can use

$$\beta^{t+1/2} = \left(1 - \frac{\alpha\lambda}{n}\right) \beta^t,$$

and $v^{t+1/2} = v^t$.

To update to w^{t+1} we can use $\beta^{t+1} = \beta^{t+1/2}$ and use

$$v^{t+1} = v^{t+1/2} - \frac{\alpha g(x_i, \beta^t v^t)}{\beta^{t+1/2}} x_i.$$

The two steps now cost $O(m)$.