CS 189 Introduction to Machine Learning
Fall 2017 gs3

1 Kernel PCA

Let $X \in \mathbb{R}^{n \times d}$ be a matrix with rows $x_1^T \dots x_n^T$, and $\phi : X \longrightarrow X'$ be a feature map with associated kernel $k(x,y) = \langle \phi(x_1), \phi(x_2) \rangle$. We will attempt to perform kernel PCA on X using the feature mapping ϕ. For this problem, assume that the data is already centered.

(a) Let $\Sigma = \frac{1}{n} \sum_{i=1}^n x_i x_i^T$ be the sample covariance matrix. Recall that performing PCA involves solving for the eigenvectors and eigenvalues of Σ.

Show if v is a solution (an eigenvector of Σ, e.g. $\Sigma v = \lambda v$), then v is in the range of X^T, or $v = X^T \alpha$ for some α.

Solution:

$$\Sigma v = \frac{1}{n} \sum_{i=1}^n (x_i x_i^T) v = \sum_{i=1}^n \frac{1}{n} x_i \langle v, x_i \rangle = X^T \alpha$$

where $\alpha_i = \frac{1}{n} \langle v, x_i \rangle$.

(b) Show if (α, λ) is a solution to $XX^T \alpha = \lambda \alpha$, then $(v = X^T \alpha, \lambda)$ solves $X^T X v = \lambda v$.

Solution:

If (α, λ) solves $XX^T \alpha = \lambda \alpha$, then $Xv = \lambda \alpha$ and by left multiplying each side by X^T, we have $X^T X v = \lambda X^T \alpha = \lambda v$.

(c) Kernel PCA solves $\phi(\Sigma) v = \lambda v$, where $\phi(\Sigma) = \frac{1}{n} \sum_{i=1}^n \phi(x_i) \phi(x_i)^T$. Explain how to find all λ satisfying this equation without explicitly computing $\phi(x)$.

Solution:

By the second problem, we know we can just solve $\phi(X)\phi(X)^T \alpha = \lambda \alpha$ where $\phi(X)$ is just a matrix with rows $\phi(x_i)^T$. Notice that $(\phi(X)\phi(X)^T)_{i,j} = k(x_i, x_j) = K_{i,j}$, where K is our kernel matrix. Thus we can compute our kernel matrix K, and then solve $K\alpha = \lambda \alpha$.

(d) In regular PCA, the projection coefficient of x onto a principal component v is $\langle x, v \rangle$. In feature space, the coefficient is $\langle \phi(x), v \rangle$. How do we compute this without explicitly computing $\phi(x)$?

Solution:

$$\begin{aligned}
\langle v, \phi(x) \rangle &= \langle \phi(X)^T \alpha, \phi(x) \rangle \\
&= \alpha^T \phi(X) \phi(x) \\
&= \alpha^T \begin{bmatrix} \langle \phi(x_1), \phi(x) \rangle \\ \vdots \\ \langle \phi(x_n), \phi(x) \rangle \end{bmatrix} \\
&= \alpha^T \begin{bmatrix} k(x_1, x) \\ \vdots \\ k(x_n, x) \end{bmatrix}
\end{aligned}$$

With this formulation, we can compute $k(x_1,x) \dots k(x_n,x)$ instead of computing $\phi(x)$.

(e) To get the correct projection coefficient in regular PCA, we always normalize eigenvectors v so that $\langle x, v \rangle$ gives the correct coefficient. How can we equivalently ensure proper normalization in our kernel PCA?

Solution:

We have

$$\langle v, v \rangle = \langle X^T \alpha, X^T \alpha \rangle = \alpha^T X X^T \alpha = \alpha^T (\lambda \alpha) = \lambda \langle \alpha, \alpha \rangle.$$

Thus, to ensure $\langle v, v \rangle = 1$, we can scale α such that $\langle \alpha, \alpha \rangle = \frac{1}{\lambda}$.

2 Convnet

(a) Do it! (Quick Check)

(a) Suppose the input to our CONV layer is the following 5×5 binary image:

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

which uses the following 3×3 filter with stride 1:

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & -1 \end{bmatrix}$$

find the output of this layer for the input. How many parameters does this layer have?

Solution:

$$\begin{bmatrix} -1 & 2 & 1 \\ 1 & 0 & -1 \\ 0 & 0 & 2 \end{bmatrix}$$

The number of parameters is the size of our filter, which is 9. Remember that this filter is learnt, and that what's in the filter are simply learnt parameters.

(b) Now suppose we have a max pooling layer with kernel size 2 and stride 2 with the following as its input:

$$\begin{bmatrix} 1 & 5 & 8 & -1 \\ 0 & 4 & 1 & 3 \\ 9 & 8 & 1 & 2 \\ 1 & 5 & 3 & 3 \end{bmatrix}$$

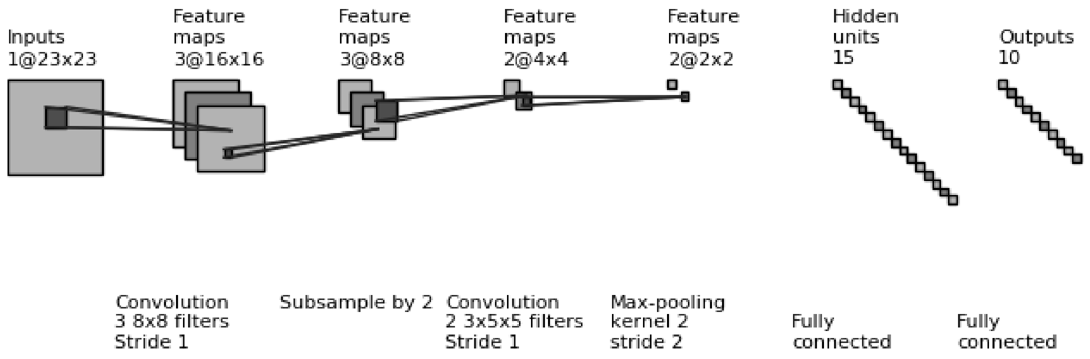
find the output to this layer. How many parameters does this layer have?

Solution:

$$\begin{bmatrix} 5 & 8 \\ 9 & 3 \end{bmatrix}$$

this layer has no parameters. Max pooling layers don't have any parameters.

(b) We have a convolutional network that takes a 23x23 image as its input and returns a 10 outputs. Assume we do not pad the image with zeros and there are no bias terms.



How many parameters are there in each layer?

Solution: There are *no* parameters in the subsampling and pooling layer.

In the first convolution layer, we have three filters. Each filter maps a 23x23 matrix to a 16x16 matrix. Since the filter has stride 1, then the width and length of the filter must be $23-16+1=8$. This gives us $3\times8\times8=192$ parameters.

In the second convolution layer, we have two filters that map the three 8x8 matrices to a 4x4 matrix. Notice that we are using a 3d filter where one of the dimensions of the filter is equal to the number of matrices in the second layer. The other two dimensions are of size $8-4+1=5$. So each filter size is $3\times5\times5=75$, so we get $2\times3\times5\times5=150$ parameters.

In the hidden layer, we map our two 2x2 matrices to 15 hidden nodes. So there are $2\times2\times2\times15=120$ parameters in the hidden layer.

In the output layer, there is a parameters from each hidden node to each output node, which is $15\times10=150$ parameters.

There are $192+150+120+150=612$ parameters in total.

(c) Suppose we have a deep convolutional neural network (a conv-net with many layers) that, given a portrait of a person, predicts whether the person in an image is Professor Sahai, or not Professor Sahai. Assume we train this classifier only with full body images of both Professor Sahai and other people. Classify the following features as “early” if they are features that are more likely to be learned in the early layers of the network, “late” if they are features that are more likely to be learned in the late layers of the network or “none” if they aren’t likely to be learned in the network.

(a) Edges

(b) This image contains glasses

(c) This image contains long hair

(d) The image contains stripes

(e) The person in the image is teaching 189

(f) The person in the image is Stella Yu

Solution:

(a) Edges - Early

(b) This image contains glasses - Late

(c) This image contains long hair - Late

(d) The image contains stripes - Early

(e) The person in the image is teaching 189 - None; our network learns features of a portrait, not high level concepts that indicate something about the person in the image

(f) The person in the image is Stella Yu - None; our network predicts if Anant Sahai or not, doesn't do anything about other professors

(d) Convolutional filters: For the following problems, assume that there is zero padding.

a) Suppose you had a n by n filter

$$\frac{1}{n^2}\begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ \dots & \dots & \dots & \dots \\ 1 & 1 & \dots & 1 \end{bmatrix}$$

What is the effect of increasing n on the result of the convolution of the filter with the input image? **Solution:** Increasing n will blur the image more

b) Suppose you wanted your convolutional layer to learn vertical edge features. Using a 3 by 3 matrix as your filter, what would the parameters inside that filter be? *Hint: we want the result of the filter to be large if there is a vertical line, and small otherwise. We also want to be able to mitigate horizontal lines. Try drawing out small examples of vertical lines/horizontal lines and think about what the results should be.* **Solution:**

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

c) Suppose you had a 3 by 3 filter

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

What feature(s) does this filter learn? **Solution:** This filter learns edge features at any orientation.

d) Using the filter in part (c), give an approach for sharpening an image. **Solution:** Since we know how to find edges using the filter in part (c), we can take the result of convolving the filter in part (c) and add it back to the image to effectively emphasize edges in the original image.

3 Decision Tree

(a) Entropy Justification

Entropy, in physical sciences, is defined as $k_B \ln W$, where k_B is Boltzmann's constant and W is the multiplicity of the coarse-grained state, or the number of ways the coarse-grained state can be arranged in. This is true under the assumption that each microstate that coarse state can be in has equal probability.

In this scenario, we have sequences of coin tosses with bias p.

(a) Boltzmann's definition of entropy is $-k_B \sum_v p(v)\ln(p(v))$. Show that in the case that each microstate v has the same probability, that this is reduced to the form given above.

(b) What is the probability of a sequence HHHHHHHH? How many ways can we have a sequence with 8 heads?

(c) What is the probability of a sequence HTHTHTHT? How many ways can we have a sequence with 4 heads?

(d) If we characterize a sequence by how many heads appeared, which case has a larger multiplicty: 4 heads, or 0 heads?

(e) We can see that a coarse-grained state with a large multiplicity has more inherent uncertainty, meaning that we are less certain about its exact specifications than we are about a coarse-grained state with a small multiplicity. In the context of decision trees, why is this useful for splitting?

Solution:

(a) If each microstate has equal probability, and there are W microstates, then each microstate has probability $\frac{1}{W}$. Thus, $-k_B \sum_v \frac{1}{W}\ln(\frac{1}{W}) = k_B \ln(W) \sum_v \frac{1}{W} = k_B \ln(W)$

(b) p^8. There is only 1 way to have a sequence with 8 heads.

(c) $p^4(1-p)^4$. There are $\binom{8}{4}$ ways to have a sequence with 4 heads. Note that when $p = 0.5$, the probabilities of every sequence are the same.

(d) The set of sequences with 4 heads has a larger multiplicity, since there are many more ways to have this sequence.

(e) A split rule that decreases the total multiplicity of the coarse-grained state is very useful because it removes uncertainty from our coarse-grained state, and in particular, it allows us to be more confident in our classifications.

(b) Decision Trees

We have a dataset with two features, labeled with classes $+$ and $-$, represented as tuples $(f_1, f_2, +/-)$:

$$\{(0,0,-),(1,0,-),(2,0,-),(0,1,+),(1,1,-),(2,1,+),(0,2,+),(1,2,+),(2,2,-)\}$$

Let L_1 be the number of class $+$ points in the left node, L_2 be the number of class $-$ points in the left node, etc. Here, we will use the splitting rule that splits on the feature that maximizes the value of $|L_1 - R_1| \cdot |L_2 - R_2|$. What is the decision tree that is generated using this splitting rule that correctly classifies the data given?

Solution: Using this splitting rule, we obtain the following set of rules:

```
if f_2 < 1:
    return -
else:
    if f_2 < 2:
        if f_1 < 2:
            return +
        else:
            return -
    else:
        if f_1 < 1:
            return +
        else:
            if f_1 < 2:
                return -
            else:
                return +
```

(c) More decision trees

We will now use a different cost function J(S). We will allow J(S) to be defined as the number of points not in a certain class C. Let's assume that we have two different distribution of points. Set 1 is {AX: 10, AY: 9, BX: 10, BY: 1}. Set 2 is {AX: 10, AY: 5, BX: 10, BY: 5}. Assume that the first split is on the first letter and the second split is on the second letter in the pair.

(a) According to J(S), what is the cost for both of the splits? Is this a good cost function?

(b) Compare the result of computing the split based on the cost function J(s) given above vs a split determined on entropy and surprise.

Solution:

(a) In set 1 we note that $J(S) = 9 + 1 = 10$. For set 2 we also have $J(S) = 5 + 5 = 10$. Therefore this is a poor cost function since the second data split represents a better split of the data points into 2 sets as compared to the first function. The cost function fails to recognize this difference.

(b) According to the original cost function we note that both of the cost functions have identical splits for each side.
Set 1:

$$\begin{aligned}
H(S1) &= -\frac{20}{30}\ln\frac{20}{30} - \frac{10}{30}\ln\frac{10}{30} = 0.918 \\
H(S2) &= -\frac{10}{19}\ln\frac{10}{19} - \frac{9}{19}\ln\frac{9}{19} = 0.998 \\
H(S3) &= -\frac{10}{11}\ln\frac{10}{11} - \frac{1}{11}\ln\frac{1}{11} = 0.439 \\
h_{\text{after}} &= \frac{|S_l|H(S_l) + |S_r|H(S_r)}{|S_l| + |S_r|} = 0.793 \\
\text{info gain} &= 0.125
\end{aligned}$$

Set 2:

$$\begin{aligned}
H(S1) &= -\frac{20}{30}\ln\frac{20}{30} - \frac{10}{30}\ln\frac{10}{30} = .918 \\
H(S2) &= -\frac{10}{15}\ln\frac{10}{15} - \frac{10}{15}\ln\frac{10}{15} = 0.23 \\
H(S3) &= -\frac{5}{15}\ln\frac{5}{15} - \frac{5}{15}\ln\frac{5}{15} = 0.23 \\
h_{\text{after}} &= \frac{|S_l|H(S_l) + |S_r|H(S_r)}{|S_l| + |S_r|} = .46 \\
\text{info gain} &= 0.458
\end{aligned}$$

Thus we note that the split in which the data is more equally distributed has a larger info gain as expected.

(d) Random Forests
Now we will consider a situation where we are presented with n training points in a feature space of d dimensions. Assume that we are trying to create a random forest with t trees, where each tree contains exactly k internal nodes, and we only select f out of the d features at each node. Answer the following questions

(a) Suppose that there is a particularly strong predictor-feature for the training points we are currently working with. What is the probability that this feature will not appear in any of the splits of the internal nodes?

(b) What happens to this probability as f approaches d?

(c) What happens to this probability as f approaches 1?

(d) What is the tradeoff between picking more or less features at each node in the random forest?

Solution:

(a) There is a total of k internal nodes, and at each internal node, the probability that the strong predictor will never get chosen for one tree is $(\prod_{i=0}^{f}(1-\frac{1}{d-i}))^k$. Since there is a total of t trees, the probability is $(\prod_{i=0}^{f}(1-\frac{1}{d-i}))^{kt}$

(b) The probability would approach 0

(c) The probability would become higher

(d) The more features you choose at each node, the higher the chances that you will pick the strong predictor-feature. This would result in many trees that all split on the same feature, which would defeat the purpose of having a random forest with different decision trees