COMP 632: Assignment 2

Due on Wednesday, February 18 2015

Presented to Dr. Doina Precup

Geoffrey Stanley
Student ID: 260645907

Question 1

A)

For a function to be considered a kernel function the kernel matrix defined as $K_{ij} = K(x_i, x_j)$ must have two properties:

1. be symmetric
2. be positive semidefinite

As such, a Kernel matrix must abide by the following:

$$K_{ij} = K_{ji} \tag{1}$$

$$z^T K z \geq 0 \tag{2}$$

Where z is an arbitrary vector.

A kernel function is also one that can be expressed as a dot product of the feature vectors of the instances:

$$K(x, z) = \phi(x) \cdot \phi(x) \tag{3}$$

where ϕ is a feature mapping of input features to a vector space. Further more, as described by Bishop(2006) p.296 a kernel function can be a construction of kernel functions such that

$$k(x, z) = k_1(x, z) + k_2(x, z) \tag{4}$$

In this particular case it would be useful to decompose our K_l kernel function into:

$$k_l(x, z) = k_1(x, z) + k_2(x, z) + ... + k_l(x, z) \tag{5}$$

where the components of the decomposition evaluate the similarity of a particular character length such that k_1"ar", "ark") = 1 and k_2("ar", "ark") = 2. Now, in order to show that K_l is a kernel function, we are left with having to create a feature mapping onto some vector where dot products of those vector would result in the similarity required.

For simplicity let's define our alphabet as being a, b, c and our mapping to be onto a three dimensional vector representing our alphabet. Now we can map an input string such as "ab" onto a vector $x = [1, 1, 0]$. Given another string "bc" we can create $y = [0, 1, 1]$ where $x^T y = 1$. Expanding this mapping into longer character combinations we can now define our vector space as being all combination of character sequences of length 1 through l.

We now have multiple functions that can be expressed in terms of a dot product of vectors whose sum is equal to our original K_l function and have thus demonstrated that it is a kernel function.

B)

Two differences will be noticeable between the kernel matrices K_l and K_{l+1}. The first will be that the values of points K_ij in the matrix where the similarities between words are of more importance will increase by at least 1. Secondly, the points along the diagonal of the matrix where $i = j$ will necessarily increase there the length of the words is greater then or equal to $l + 1$. As such, the result will be that the values along the diagonal of the matrix will necessarily increase by at least the same amount as other values in their respective rows and columns.

Therefor, the tendency of the matrix is to become more diagonal in nature as l increases. Which will result in a model that tends more towards over fitting.

In order to avoid this problem a methodology such as five fold cross validation should be used in order to identify an optimum value for l. The objective would be to identify at which value of l the algorithm has a tendency of generating smaller error during training then during testing and to use an l that is smaller in value then this.

C)

Yes. Words with a longer length will have a tendency of having larger values of for any given l of the kernel function. For example, $K_2(\text{"}bird\text{"}, \text{"}bird\text{"}) = 7$ and $K_2(\text{"}apple\text{"}, \text{"}apple\text{"}) = 9$. As kernel functions are meant to be a measure of similarity between vectors this result is misleading.

A potential adjustment to this correlation would be to divide the result by some value to turn the comparison into a percentage value. I believe an appropriate adjustment would be to adjust the kernel function as follows:

$$K_l(x,z) = \frac{K_l(x,z)}{max(K_\infty(x,x), K_\infty(z,z))} \tag{6}$$

This would ensure that a comparison of any string with itself would have a maximum value of 1. Further more, for any string, the kernel would return a comparison value that would be adjusted with regards to the string that it is most similar to, itself.

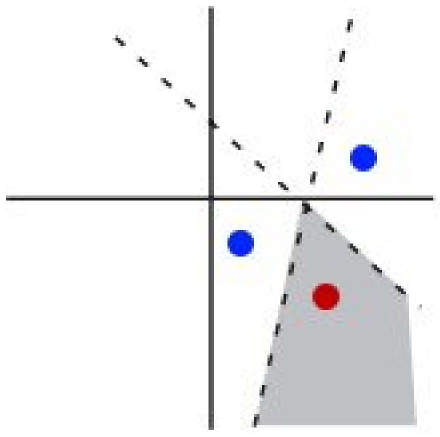
D)

Yes. One would be able to use the same methodology I used above to demonstrate this. In order to implement such an approach one would simply have to alter the string to vector mapping definition used.

More precisely, instead of associating a value of 1 to all character sequences one could associate different values according to their desired importance. For example, when a string contains the sequence "ar" the mapping could put a value of 2 to the dimension associated to this sequence but 1 to all others.

Question 2

A)



B)

The VC-dimension of this hypothesis class is 5. This is because it can shatter all configurations of 4 points. However, it would not be able to do so for all configuration of 5 points.

C)

The VC-dimension of any type of boolean combination of 2 linear classifiers is 6. The reason is that we are not limited by conjunctions. This limitation is what prevented us from shattering all configurations of 5 points in the previous example. Now that this limitation is removed, we can successfully shatter all combinations of 5 and we can at least shatter one combination of 6 but not all. As such, the VC-dimension in this case is 6.

Question 3

A)

Given the likelihood of an example x belonging to class k as being :

$$L(k|x) = 1 - \sum_{i=1}^{K-1} h^i(x) \tag{7}$$

From Bishop (2006), the likelihood function of a multiclass logistic regression can be calculated as:

$$L(T|w_1, ..., w_k) = \prod_{n=1}^{N}\prod_{k=1}^{K} p(C_k|\phi_n)^{t_{nk}} = \prod_{n=1}^{N}\prod_{k=1}^{K} y_{nk}^{t_nk} \tag{8}$$

Because the exponent $t_{nk} = 0$ when an element is not of a particular class k the second product is equivalent to equation 7. As such, the likelihood function for a set of hypotheses and a given data set D can be expressed as :

$$L(T|w_1, ..., w_k; D) = \prod_{n=1}^{N}\prod_{k=1}^{K} y_{nk}^{t_nk} \tag{9}$$

where $y_nk = y_k(\phi_n)$.

B)

$$-\log p(T|w_1, ..., w_K) = -\sum_{n=1} N \sum_{k=1} K t_{nk} \log y_{nk} \tag{10}$$

$$\nabla_{w_j} E(w_1, ..., w_k) = \sum_{n=1}^{N} (y_{nj} - t_{nk})\phi_n \tag{11}$$

$$\nabla_{w_k}\nabla_{w_j} E(w_1, ..., w_k) = \sum_{n=1}^{N} y_n k (I_{kj} - y_{nj})\phi_n\phi_n^T \tag{12}$$

C)

Question 4

A)

	Folds				
	1	2	3	4	5
log L Train	-0.410	-0.408	-0.409	-0.433	-0.422
log L Test	-0.654	-0.558	-0.595	-0.442	-0.526
Training Accuracy	0.81%	0.81%	0.83%	0.81%	0.80%
Testing Accuracy	0.69%	0.69%	0.72%	0.87%	0.71%

B)

	Folds				
	1	2	3	4	5
log L Train	-0.049	-0.067	-0.026	-0.028	-0.183
log L Test	-0.333	-0.355	-0.521	-0.376	-0.239
Training Accuracy	0.65%	0.65%	0.64%	0.65%	0.65%
Testing Accuracy	0.62%	0.62%	0.59%	0.54%	0.68%

C)

	Folds				
	1	2	3	4	5
log L Train	-0.235	-0.212	-0.174	-0.184	-0.226
log L Test	-0.530	-0.611	-0.803	-0.633	-0.536
Training Accuracy	0.83%	0.84%	0.83%	0.81%	0.83%
Testing Accuracy	0.72%	0.67%	0.69%	0.85%	0.74%

D)

Defining the sigmoid function as being:

$$\sigma(a) = \frac{1}{1+e^{-a}} \tag{13}$$

We can now express the standard logistic regression as:

$$P(Y=1|X) = \sigma(w^T X + w_0) = \sigma(w_0 + \sum_{i=1}^{n} w_i X_i) \tag{14}$$

$$P(Y=0|X) = 1 - P(Y=1|X) \tag{15}$$

Using a function ϕ mapping X from a low to a high dimension space we obtain the following logistic regression:

$$P(Y=1|\phi(X)) = \sigma(\sum_{i=1}^{K} w_i \phi(X)_i) \tag{16}$$

where K is the dimension of the new higher dimension space. Using the kernel trick:

$$P(Y=1|\phi(X)) = \sigma(\sum_{i=1}^{n} \alpha_i K(X_i, X)) \tag{17}$$

Combining equation 16, 17 and 19 we obtain the likelihood of α

$$\log L(\alpha) = \sum_{i=1}^{n} y_i \log \sigma(\sum_{j=1}^{n} \alpha_i K(X_j, X)) - (1-y_i)\log(1+\sigma(\sum_{j=1}^{n} \alpha_i K(X_j, X))) \tag{18}$$

Defining $h(X) = \sigma(\sum_{i=1}^{n} \alpha_i K(X_i, X))$ the gradient of $\log L(w) wrt \alpha$

$$\begin{aligned}\nabla \log L(\alpha) &= \sum_i y_i \frac{1}{h(X_i)} h(X_i)(1-h(X_i)) \sum_{j=1}^{n} K(X_j, X_i) \\ &+ (1-y_i)\frac{1}{1-h(X_i)} h(X_i)(1-h(X_i)) \sum_{j=1}^{n} K(X_j, X_i)(-1) \\ &= \sum_i \sum_{j=1}^{n} K(X_j, X_i)(y_i - y_i h(X_i) - h(X_i) + y_i h(X_i)) \\ &= \sum_i (y_i - h(X_i)) \sum_{j=1}^{n} K(X_j, X_i) \\ &= K^T(Y-\hat{Y})\end{aligned} \tag{19}$$

Therefore the gradient descent algorithm becomes:

$$\alpha_{t+1} = \alpha_t + \gamma K^T(Y-\hat{Y}) \tag{20}$$

	Folds				
	1	2	3	4	5
log L Train	-0.014	-0.014	-0.014	-0.014	-0.014
log L Test	-0.655	-0.662	-0.660	-0.665	-0.677
Training Accuracy	1.00%	1.00%	1.00%	1.00%	1.00%
Testing Accuracy	0.72%	0.72%	0.67%	0.72%	0.76%

E)

Five fold cross validation

Five fold cross validation was facilitated by the numpy library of Python's Scipy package as well as some methods from the ScikitLearn package.

The first step was to load the two dat files into numpy arrays. Following this ScikitLearn's shuffle method was used with a random seed of 125. Once the data was properly shuffled it was split into 5 sections with numpy's split method. This allowed for the grouping of data into training and testing subsets. In the first fold the first split index was used as the testing set and in the second fold, the second index, and so on and so forth. The remaining 4 indices were used as the training data.

This procedure was used to train and test logistic regression, Gaussian discriminant analysis and kernelized logistic regression.

Logistic Regression

For logistic regression ScikitLearn's LogisticRegression class was used. The model was fitted using the training data from the five fold cross validation procedure.

From there, the predict log proba method of the class was used to predict log likelihoods of both classes of the model. In order to compute the average log likelihood, the greatest of the two were added together and subsequently divided by the quantity of examples. This process was done for both the training and the testing data set.

The accuracy was caculated by counting the quantity of correctly classified targets and then dividing by the amount of examples. This was done for both the training and testing data sets.

Gaussian Discriminant Analysis

Gaussian discriminant analysis was implement with customized code. The modelizing step involved computing a prior, mean and covariance matrix for both example with 0 targets and 1 targets. Prior was calculated by dividing the amount of targets of type 0 and 1 by the total amount of examples. Mean was calculated using numpy's mean method and the covariance matrix was computed using numpy's cov method. To compute the diagonal matrix, a full matrix was computed at first, and subsequently all points where $i \neq j$ were set to 0.

To calculate the log likelihood of features the following formula was used:

$$\log L(X) = \log \pi + X^T \sigma^{-1} \mu - \frac{1}{2} \mu^T \sigma^{-1} \mu \tag{21}$$

Once calculated, the class with the highest log likelihood was chosen as the prediction. Average log likelihood and accuracy were calculated in the same manner as with logistic regression.

Kernelized Logistic Regression

Kernelized logistic regression was implemented with customized code. The kernel used was a gaussian kernel defined by :

$$K(X, Z) = \exp\left(-\frac{\|X - Z\|^2}{2\sigma^2}\right) \tag{22}$$