CS 412 – Data Mining

Final Exam

May 7th 2021

Pouya Akbarzadeh (pa2)

Problem 1

a. We can not use the hard margin for the formulation of the SVM because there is an assumption with hard margin that the data set is linearly separable by class. In the 2 classes we have neither is linearly separable. At least without major modification of the data.

b. Yes, but we need to modify the data set. Thus, for us to be able to use SVM we need to modify our data set and project it into a higher dimension, so it is linearly separable. This would be using radial basis function. Adding a Z axis would make these classes linearly separable. Only then can we train it.

c. While SVMs are well known for how effective they are in high dimensional spaces, we need to understand that that is where number of features is greater than number of observations. However, by increasing our data set we can have what is called overfitting. This could happen since we are going from a 2D to a 6D space. Theoretically it should give us an accurate predictor, however in reality it is not likely. Thus, it would actually hurt our accuracy rather than help.

Problem 2 (I am using code to show intermediate steps)

a. The degrees of freedom can be calculated by k-1, thus it would be 9. For a given model, the individual error rates calculated in the cross-validations may be considered as different, independent samples from a probability distribution. In general, they follow a t-distribution with k − 1 degrees of freedom where, here, k = 10. (This distribution looks very similar to a normal, or Gaussian, distribution even though the functions defining the two are quite different. Both are unimodal, symmetric, and bell-shaped.) This information was found in our textbook.

b. The following was equation was found in our textbook. Using that, I got t-stat:-19.529

$$t = \frac{\overline{err}(M_1) - \overline{err}(M_2)}{\sqrt{var(M_1 - M_2)/k}},$$

Allow me to further explain the equation, and values gained from them. The variable t is the test statistic, err(M_1) is the mean of error rate of A, and err(M_2) is for B respectively. And it can be easily understood that the var(M_1-M_2) is the variance of A – B. The division of k is equivalent to number of folds we have, thus in our question it would be 10.

$$var(M_1 - M_2) = \frac{1}{k}\sum_{i=1}^{k}[err(M_1)_i - err(M_2)_i - (\overline{err}(M_1) - \overline{err}(M_2))]^2$$

Furthermore, we can calculate p using python.

```
from scipy.stats import t
import numpy as np
num = A_mean_error - B_mean_error
denum = np.sqrt(np.var(A - B)/k)
pval = (1-t.cdf(abs(num/denum), k-1)) * 2
print(pval)
```

Where the values used above are derived from

```
A = np.array([0.908, 0.962, 0.878, 0.956, 0.939, 0.955, 0.944, 0.933, 0.881, 0.949])
B = np.array([0.449, 0.585, 0.381, 0.433, 0.475, 0.430, 0.520, 0.590, 0.565, 0.443])
A_new = np.array([0.908, 0.962, 0.878, 0.956, 0.939, 0.955, 0.944, 0.933, 0.881, 0.949])
B_new = np.array([0.968, 1.000, 0.950, 0.994, 0.989, 0.989, 1.000, 0.994, 0.966, 0.966])

A_error = 1 - A
B_error = 1 - B

A_mean_error = np.mean(A_error)
B_mean_error = np.mean(B_error)
```

Please note that k=10 for us.

The value printed by the code above was 1.12×10^{-8}. We can see that the p value is less than 0.05, thus we can assume that out of our algorithms, one is much much better.

c. Just like the part above, we calculate the t-stat to be 8.532. We use the same code shown above to calculate the p-value. This time we get 1.32×10^{-5}. Again, we can see that the p value is less than 0.05, thus we can assume that out of our algorithms, one is much much better.

Problem 3

a. We can see that our w_j is defined as

$$w_j^{\text{new}} = w_j + \eta g'(a^i)(y^i - \hat{y}^i)x_j^i$$

And based on

where $a^i = \mathbf{w}^T\mathbf{x}^i$, and the gradient of the ReLU function is

$$g'(a^i) = \begin{cases} 1\ , & \text{if } a^i \geq 0\ , \\ 0\ , & \text{otherwise}\ . \end{cases}$$

We can derive how the parameter is updated.
With SGD, we update the weights incrementally for each training sample.
We see that $g'a(y^i - \dot{y}^i)$ gives us the steepness of the gradient at a randomly selected point. Since the weights are updated in the direction of the gradient, and the gradient is defined by the partial derivatives of the error function, we need to take the partial derivative in terms of w. Due to my poor math skills and time we now announce partial derivative PD_w. Thus, our new equation for w_{new}^j will be the following.

$$w_{new}^j = w_j + \eta g * PD_w$$

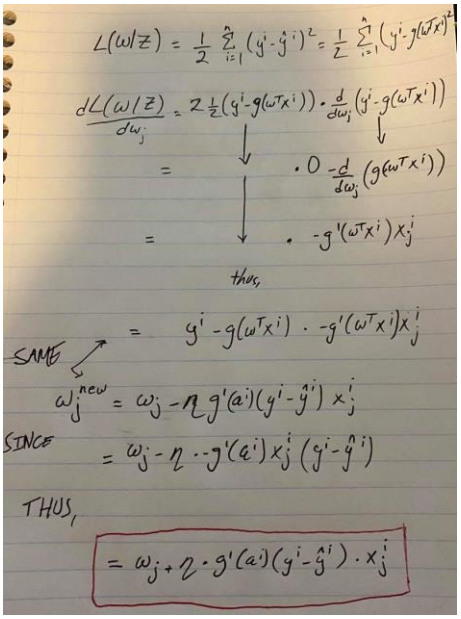
Looking at the equation of L(w|Z) we can take a partial derivative of it in respect to w. This will allows us to set up the gradient descent alg.

$$L(\mathbf{w}|\mathcal{Z}) = \frac{1}{2}\sum_{i=1}^{n}(y^i - \hat{y}^i)^2 = \frac{1}{2}\sum_{i=1}^{n}(y^i - g(\mathbf{w}^T\mathbf{x}^i))^2\ .$$

The derivative should result in the following

$$\frac{\mathrm{d}}{dw}(\mathrm{L}(\mathrm{w}|\mathrm{z}) = \sum_{i=1}^{n}(y^i - g(w^T x^i)) * PD_w(y^i - g(w^T x^i))$$

The actual steps are handwritten as seen in the picture below. (Sorry, I have 2 other exams)



We can see that taking the derivative allowed us to get the same equation.

b. Instead of ReLU function, we will now use a linear transfer function. I believe based on initial explanation given in part a. The final answer would be.

and thus

$$g'(a^i) = \begin{cases} 1, & \text{if } a^i \geq 0 \ , \\ 0 \ , & \text{otherwise} \ . \end{cases}$$

This allows for linear shift per iteration aka when the steps change.

Problem 4

a. K-medoids algorithm is that breaks the dataset up into groups and tries to minimize the distance between the points (partitioning). K-medoids is a more robust version of k-means and less vulnerable to noise and outliers. This is done in a manner of a cluster and one point is in the center of that cluster. K-medoids are used with arbitrary dissimilarity measures.
The algorithm has 2 main parts to it
To put our code in words (pseudo code)
The first part is the build phase where we

1. Select k objects (these objects will become our medoids)
2. Calculate the dissimilarity matrix
3. Assign every object to its closest medoid

After we are done with the build phase, we move to the swap phase where we

4. Go through each cluster and find if there are any k objects that will decrease the average dissimilarity coefficient
5. IF there is such object in cluster, select it as the medoid
6. IF at least one medoid has changed, we need to go back to step 3
7. ELSE we finish the algorithm

b. The time complexity per iteration is $\mathbf{O(k(n-k)^2)}$, however due to poor implementation and recomputing the entire cost function every time we can have a time complexity per iteration of $O(k^2n^2)$. To achieve the time complexity of $O(k(n-k)^2)$ only the change in cost needs to be recalculated. We can decrease runtime even further to $O(n^2)$. The reason the time complexity is $O(k(n-k)^2)$ is due to the loop configuration of for each non-medoid point (n-k) , its squared since we have 2 loops, then we have to consider that we have to go though every medoid which adds a factor of k, thus $O(k(n-k)^2)$.

c. While k-medoids is very similar to k-means, due to their partitioning approach there are some differences. Furthermore k-median clustering is a verification version of k-means where we don't cluster the mean for each cluster to figure out the centroid, rather the median. Now this change has the impact of minimizing the error over all cluster with the respect to the 1-norm distance metric. It is important to note that k-means in the other has the squared 2-norm distance metric.
Furthermore, we can say that k-means minimizes the within cluster variance which is equivalent to squared Euclidean distances while k-medians minimizes the absolute deviations which is equivalent to Manhattan distance.

Now moving on to explain the demand for computation. Both K-medians and k-medoids are far more computationally expensive than k-means. This is because of the way their algorithms are set up. For example, k-medoids needs to compute the pairwise similarity of matrices between data points which is considered expensive for larger data sets. Now, to find median, the coordinates of cluster (data) need to be sorted which can be more costly than simply calculating the mean. It is important to note that these differences become more and more apparent as size of data increases.