Data Integration on High-Difficulty Binary Classification

Julia Finch, Jesse Hellemn, Kentaro Hoffman, Zhe Zhang

April 29, 2017

Abstract

This paper explores the effectiveness of the data integration technique Generalized Multiple Kernel Learning (GMKL) on high-difficuly binary classification data. GMKL that integrates learned kernels from two disparate data sources is systematically compared to performing GMKL on the two sources naively concatenated as well as standard classification algorithms that are performed on the concatenated sources. The variables that are systematically varied are the number of informative dimensions in each source, the relative information provided by each source, and the number of useless noise dimensions added to each source. It is found that GMKL consistenly outperforms its competitors. This paper provides evidence that data integration techniques, specifically GMKL, have the ability to drastically improve upon the performace of naive concatenation.

Introduction

For classification tasks in many fields, especially in medicine and the social sciences, it is increasingly common for data to come from multiple disparate data sources. For example, a sociologist might be interested in predicting future income using two different sources, one on family environment and one on school environment (we will use "source" to refer to a single coherent dataset). For the best analysis and classification results, all data sources should be taken into account. However, most current machine learning classification techniques have been developed for only single dataset inputs, and it is not obvious how to best adapt these techniques to multi-source data.

A naive approach is to concatenate the sources together into one large feature matrix, essentially treating all of the data as a single incoherent source. Although simple, this method throws away information about the separate sources and forces data analysis and classification techniques to treat all of the sources in the same way. Various data integration techniques have been proposed to more cleverly and effectively combine multiple sources. Unfortunately, these techniques have been poorly tested, and there has been no systematic evaluation of their effectiveness.

This paper evaluates one such specialized data integration techniques, Generalized Metric Kernel Learning (GMKL), against traditional classifiers that use concatenated data. We simulate many 2-source datasets with a variety of properties for these comparative tests. We will use the term "classifier" to refer to both specialized data integration techniques such as GMKL as well as naive methods that first concatenate all sources together.

Data Generation

Criteria of Good Simulated Data

In order to understand the behavior of the classifiers on 2-source data as properties of the data are varied, we systematically created data to satisfy all of the criteria in . These criteria allow us to test all the classifiers fairly against each other on consistent benchmarks.

Criteria of Simulated Data

1. The dataset consists of two separate sources.
2. The two classes are separated by a true decision boundary that is known and calculable.
   - or the two classes can be specified to overlap with percentage p, where $p = 0$ leads to no overlap and $p = 50$ leads to complete overlap
3. The decision boundary's complexity (and thus difficulty) can be parametrized and controlled.
4. The reliability of each source can be specified.
5. The noisiness of each source can be specified.
   - extra meaningless dimensions can be added to each source

Explanation for Data Criteria

Suppose that a classifier C only obtains 60% classification accuracy on a dataset D (with datapoints evenly split amongst 2 classes). This could be attributable to either:

- The classifier is not well suited to certain properties of dataset D
- 80% of both classes overlap with each other. The best possible strategy in this area of overlap is to guess the class with 50% accuracy. An optimal classifier will then guess 40% of the datapoints correctly and also classify the 20% of non-overlapping datapoints perfectly

Criterion 2 ensures that the latter case does not occur, so that classifier performance is attributable solely to its efficacy on certain types of data.

Criteria **??** is more difficult than it at first seems. In order to create a source with N_{useful} useful dimensions and N_{noisy} meaningless dimensions, we needed to both 1) make N_{noisy} dimensions of pure noise and 2) make N_{useful} dimensions, all of which are always useful. With a random coefficient linear model, it is impossible to verify that all N_{useful} dimensions are actually useful.

Data Generation Models

Random Coefficient Linear Model

A simple, common way to simulate data is to use a linear model with random coefficients. This model generates data by:

1. Specify the number of true latent variables K along with the number of visible variables N
2. Specify two distribution D_j^0 and D_j^1 of each latent variable z_j, $1 \leq j \leq K$, where D_j^0 is the distribution of z_j for negative classes and D_j^1 is the distribution of z_j for positive classes
3. Specify how each visible variable x_j is generated from the latent variables z_i with a formula of the form

$$x_l = \sum_{i=1}^{K} \beta_i z_i + \sum_{i=1}^{K} \sum_{j=i}^{K} \beta_{i,j} z_i z_k + \text{higher-order-interactions}$$

   of linear combinations of arbitrary functions of the latent variables

4. Specify every β in the above formula
5. For every datapoint
   (a) Pick which class the datapoint belongs to
   (b) Sample each z_j from its respective distribution for this class
   (c) Generate each visible variable x_j from its formula

This model has significant shortcomings

- It is not known how to systematically make the classification problem more or less difficult
- It is hard to know if the generated data overlaps
- It is hard to pick the βs to ensure that all of the criteria in are satisfied
- There are many distributions and formulas to specify arbitrarily
- It is hard to know how many of the generated dimensions are useful

Feed-forward Network Model

In order to satisfy all of the critera in , we created a feed-forward conditional network (figure 1). This network's process is given in Algorithm **??**.

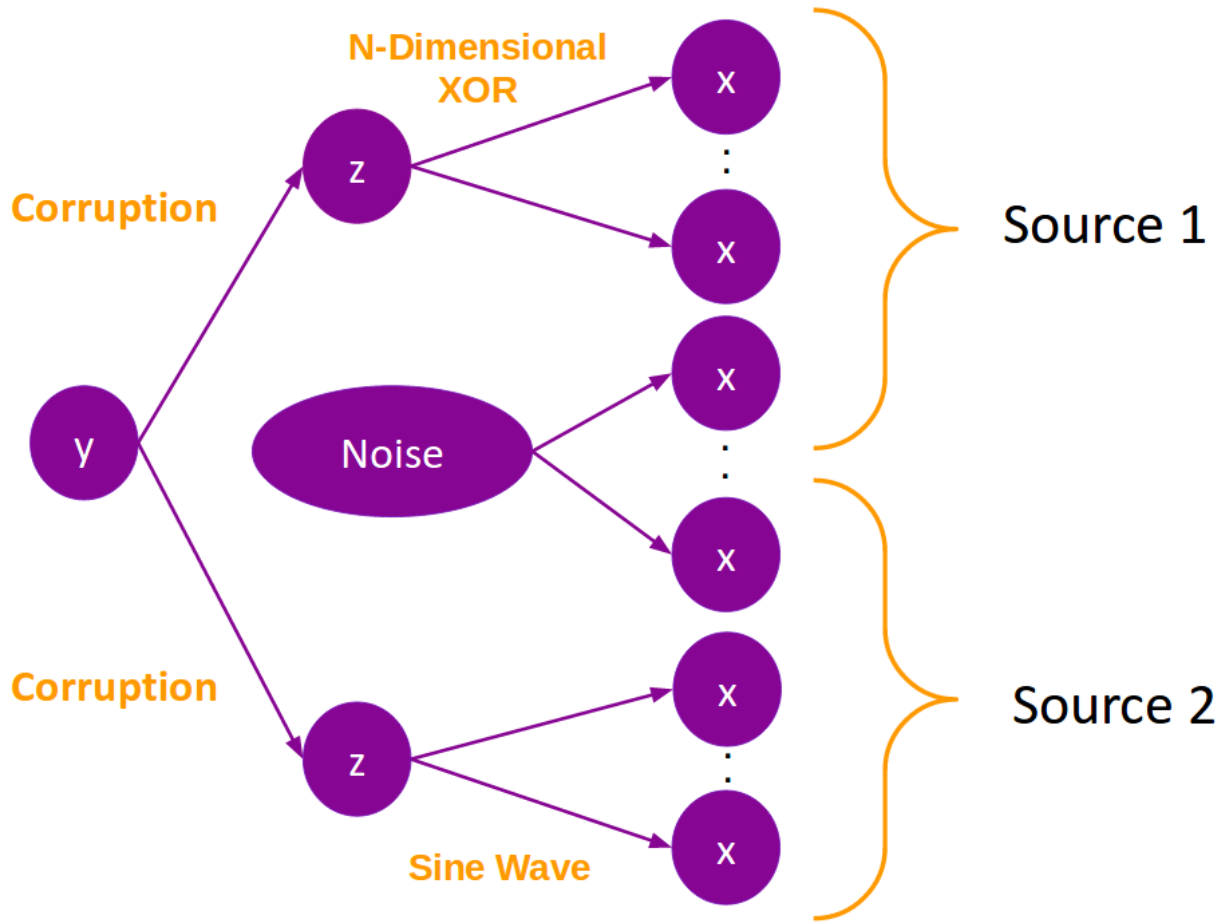


Figure 1: Caption for this network model

Algorithm 1 Network Model Data Generation Process

{Sample the y layer}
$y \leftarrow$ Bernoulli(p)
{Sample the z layer}
for all $z_i \in \{z_1, z_2\}$ **do**
  $c \leftarrow$ Bernoulli(p_i) {p_i chance to corrupt source i}
  $z_i \leftarrow c * (1 - y) + (1 - c) * y$ {If corrupting, z_i will be 0 if y is 1 and 1 if y is 0}
end for
{Sample the x layer of source 1, an N_1-dimensional XOR}
if z_1 is even **then**
  $x_1^{(1)}...x_{N_1}^{(1)} \leftarrow$ N_1-dimensional binary vector of even parity
else
  $x_1^{(1)}...x_{N_1}^{(1)} \leftarrow$ N_1-dimensional binary vector of odd parity
end if
{Sample the x layer of source 2, a k_2-period sine wave}
$x_1^{(2)} \leftarrow$ Uniform($-k_2\pi$, $k_2\pi$)
$x_2^{(2)} \leftarrow$ Uniform($-k_2\pi$, $k_2\pi$)
$x_3^{(2)} \leftarrow (x_1^{(2)} + x_2^{(2)})sin(x_1^{(2)}) + mz_2$

The above model creates two sources of data with very different types of decision boundaries.

The first source is a N-dimensional XOR, which consists of clusters of points at every corner of an N-dimensional hypercube, where each corner belongs to a different class than all of its $N-1$ closest neighboring corners. Figure 2 shows a 3D example. The complexity of the decision boundary is proportional to the dimension.

The second source is a sine wave in 3D space (figure 3); the complexity of this decision boundary is directly proportional to the period of the wave and inversely proportional to the margin m between the two classes.

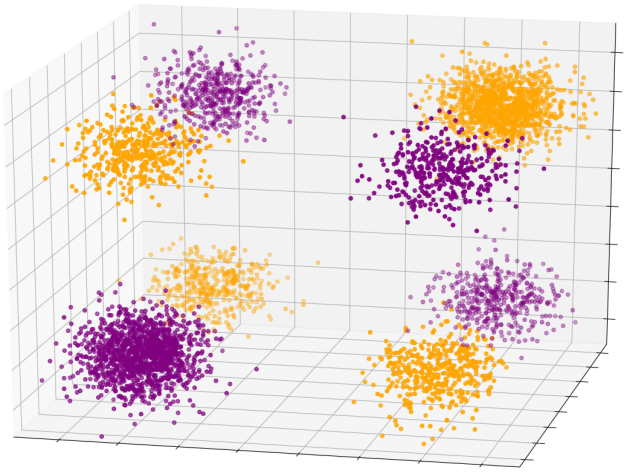


Figure 2: A 3D-XOR with equal class size and $\sigma = 0.2$ noise.

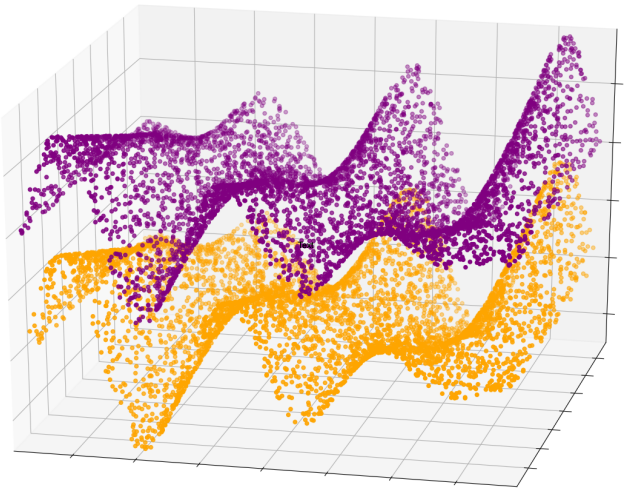


Figure 3: The 3 period sine wave created as the second source by the network example in 1, with $\sigma = 0$ noise and a margin of $m = 10$. The equation of this wave in xyz coordinates is $z = (x + y)sine(x) + mc$, where m is the margin and c is the 1 or 0 class label.

This model has several nice properties:

- For both the N-dimensional XOR and the sine wave (with small enough margin m), every dimension is necessary for perfect classification.

- The reliability of each source can be varied independently of the other.
- The data can be made more noisy (to an extent) without compromising the separability of the two classes.
- Extra meaningless dimensions can easily be added to either source.

The first experiments in this paper use a similar model to the one above, except with another XOR as the second source instead of the sinewave. The last experiments use the exact model above.

Statistical Methods

To access the potential gain from intelligent data integration, we decided to compare the classification accuracy of kernel based integration technique with commonly used vector concatenation benchmarks.

Notation

In describing different classification techniques, the following convention will be used:

- (x, y) pair denotes a feature vector x $\in \mathbb{R}^m$ and its corresponding target value $\in \{0, 1\}$.
- x_j denotes the jth feature in x.
- there are N training examples and ith training pair is represented by (x^i, y^i).

Vector Concatenation Benchmarks

Gaussian Naive Bayes

Model: The model assumes that conditioned on the value of y, every feature x_j is generated independently i.e. $P(x|y) = \prod_{j=1}^{m} P(x_j|y)$. In addition it also assumes that each $P(x_j|y) \sim \mathbf{N}(\mu_{\mathbf{j},\mathbf{y}}, \sigma_{\mathbf{y}})$. To make x, the model would compute its posterior probability $P(y|x)$ and pick the more likely case.

Properties: The formulation is clean and simple. It is fast to train and is quite resistant to extra noise dimensions. However, the excessively strong assumption of independence means that we could only fit 2nd degree decision boundaries, so it is expected to perform poorly on our highly complex XOR dataset.

Implementation: We used sklearn's Gaussian Naive Bayes classifier. No hyper-parameter tuning is required.

Random Forest

Model: The model uses a group of decision trees to make a prediction for an x. Each decision tree is based by a bootstrap sample of the training data and the splitting nodes are constrained to a random subset of all features. In this manner, the over-fit of decision tree could be controlled.

Properties: Random forest is known to be very resistant to extra noise dimensions since uninformative feature would never be used for a split in a decision tree.

Implementation: We used cross validation to select the best number of trees $\bar{t} \in \{10, 100\}$.

K-Nearest Neighbors

Model: KNN makes a prediction on an unknown data-point by the majority voting of the K-closest points i.e.

$$f(x) = sign(\sum_{x^i \in \text{K closest points}} y^i)$$

Properties: KNN is able to fit extremely complex decision boundaries. However, it often suffers from the curse of high dimensionality i.e. when the training data only occupies a tiny fraction of the high dimensional feature space. Moreover, it could not discriminate against noise dimensions since they are incorporated as part of the Euclidean distance.

Implementation: We used cross validation to select the best number of neighbors $\bar{k} \in \{1, 10\}$

RBF Support Vector Machine

Model: Similar to KNN, support vector machine makes recommendation by considering the weighted voting of a set of similar data points

$$f(x) = sign(\sum_{i=1}^{N} y^i d^i K(x, x^i)) \text{ where K(x, } x^i) = exp(-\gamma|x - x^i|^2)$$

where d^i is learned from data by finding the largest margin separating hyperplane in the projected space.

Properties: SVM can fit complex decision boundaries. Moreover, it does not suffer for the curse high dimensionality as much as KNN. Those two reasons has makes SVM the go-to algorithm for classification. However, just like KNN, it is unable to discriminate against extra noise dimensions.

Implementation: Cross-validation is performed to search for the best regularization parameter $\bar{C} \in \{0.1, 1, 10, 100\}$ and $\bar{\gamma} \in \{.01, .1, 1, 10\}$

Generalized Multiple Kernel Learning

Kernel methods allows the client to design custom kernels to incorporate prior knowledge about the dataset. In our case, we used multiple kernels to incorporate our knowledge of the data coming separate sources. This method involves several steps, and a general flowchart of the process is shown in figure 4.

Model: The final trained model is almost the same as those from SVM, expect that the final kernel function is actually learned from data. More specifically, the final kernel is a convex combination of a set of predefined kernel functions:

$$K(x, x^i) = \sum_{q=1}^{Q} \theta_q * K_q(x, x^i) \text{where } \theta_q \text{is learned from data.}$$

Formulation as Optimization Problem

$$\begin{aligned} &\underset{\theta,v,b}{\text{minimize}} && C\frac{1}{N}\sum_{i=1}^{N} L(f_{\theta,w,b}(x^i), y^i) + 1/2\sum_{q=1}^{Q} |\frac{v_q|^2}{\theta_q} \\ &\text{subject to} && \beta|\theta|_2^2 + (1-\beta)|\theta|_1 \leq 1 \end{aligned}$$

Where C is the regularization parameter, β is the elastic net parameter, L is the hinge loss function and

$$f_{\theta,w,b}(x^i) = \sum_{q=1}^{Q} w_q \phi_q(x^i)\sqrt{\theta_q} + b$$

Now if we define v by $v_q := \frac{w_q}{\sqrt{\theta_q}}$, then we have a convex optimization problem:

$$\begin{aligned} &\underset{\theta,v,b}{\text{minimize}} && C\frac{1}{N}\sum_{i=1}^{N} L(f_{\theta,w,b}(x^i), y^i) + 1/2|w|^2 \\ &\text{subject to} && \beta|\theta|_2^2 + (1-\beta)|\theta|_1 \leq 1 \\ &\text{Where} && f_{\theta,v,b}(x^i) = \sum_{q=1}^{Q} v_q \phi_q(x^i) + b \end{aligned}$$

Observe that the elastic net constrain ensures sparsity and grouping effect for the set of kernels chosen for the final model.

Optimization Algorithm

Implementation For each 'independent' data source, we constructed 10 RBF kernels with the width $\gamma \in \{2^{-3}, 2^{-2}, 2^{-1}..., 2^6\}$. Then we trained our model with the regularization parameter C fixed to 100.

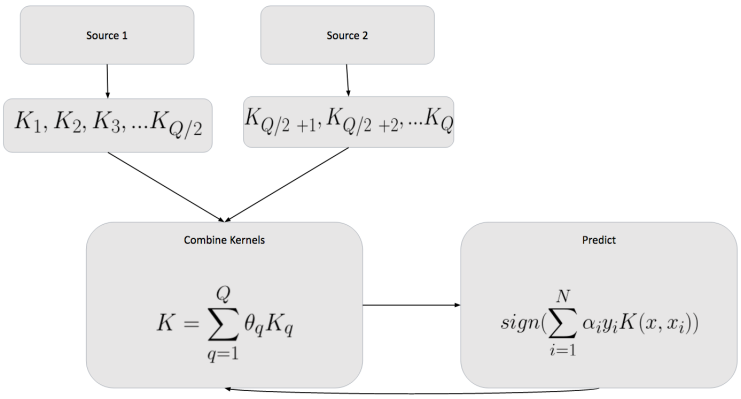


Figure 4: Flow Chart of Our Model

Experiments

Experiment 1: Data Dimension Scaling

With the prevalence of high dimensional data sets, we first would like to see if our data integration methods scale well as the dimension of the data sources increases. To this end, we generated several data sets of varying number of N_{useful} dimensions from a double-XOR feed-forward network. The network is structured such that each data source receives the parity signal without corruption from the true source, y. Then it creates an arbitrary n-dimensional XOR of correct parity, which then becomes a 2n dimensional data vector. A full set of parameters for this experiment can be seen in table 4.1. This network was chosen as it has no corruption of data sources, thus ensuring criteria 1. Second, as the dimension of the XOR sources increase, the decision boundary becomes increasingly complicated. Once the data generation process is complete, classification was done by GMKL, Concatenate + GMKL, SVM, KNN, and Random Forest.

These paramters values should be replaced with the correct variable names once section 2.2.2 is completed. Should the parameters for the classifiers be described here? or in an appendix? Or in the methods section?

Parameter	Value
Type of Data from each Data Source	[XOR, XOR]
N_{useful} (For each data source)	[2,3,4,5,6,7]
N_{noisey}	0
Number of Total Data points	5000
Ratio of Training to Testing Data	1:2
Probability of Data Source Signal Corruption	[0,0]
I must be missing some more parameters	

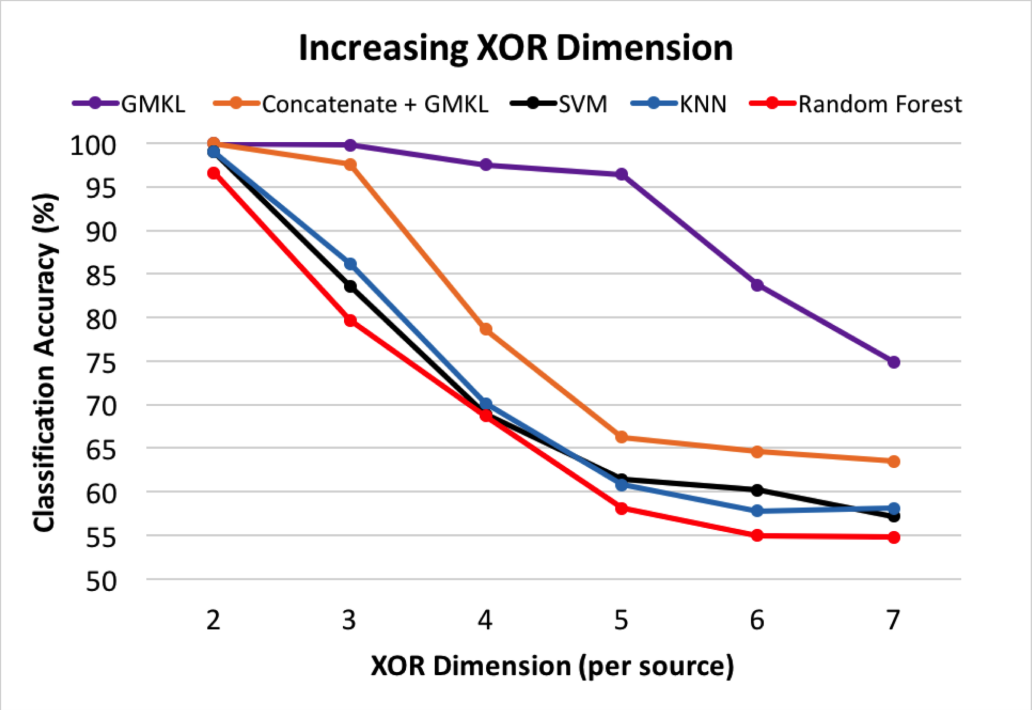


Figure 5: **PUT A CAPTION OR PARAMS OR SOMETHING HERE**

From figure **??** we can see that as number of XOR dimensions increases, the classification accuracy of all of the classifiers decreases. This makes sense as the dimensionality of the data is increased without a corresponding increase in the number of data points, making the classifiers suffer from the curse of dimensionality. However, while all classifiers have decreasing accuracy, we can see that GMKL performs much better than the other classifiers with nearly a 35 percent better classification accuracy compared to the classical SVM at at 5 dimensions. In fact, there is a very intriguing pattern on display here as the the GMKL at 2n dimensions seems to be performing about as well as the classical SVM at n dimensions. This seems to indicate that maybe doing classification on the data sources as separate entities is highly desirabel for complicated classification problems.

GMKL Dimension	Accuracy	SVM Dimension	Accuracy	Difference
2	a	4	a	
3	a	6	a	

Experiment 2: Corrupted XOR Sources

In order to corrupt the sources, a Bernoulli experiment is performed for each source with a certain probability that the binomial indicator is flipped to 0 or 1 (whichever number it is currently not). A probability p_1 is assigned to the first source and a probability p_2 is assigned to the second source. These probabilities range

from 0, meaning no corruption, to 0.5, meaning complete corruption. For example, say the true value of y is 0, p_1 is 0, and p_2 is 0.5. Then the data that stems from the first source accurately represents the value of y which is 0. The data that stems from the second source will accurately represent the value of y in 50 percent of the samples, but the other 50 percent of the samples it flips to represent 1. This means that all of the information is coming from the first source while the second source tells us nothing about the true value of y.

One variable that we decided to vary is the relative importance of the disparate sources. The purpose of this experiment is to evaluate how the various classifiers perform when one source is more important than the other. We want to see which classifers are able to identify the important sources/features.

In order to corrupt the sources, a Bernoulli experiment is performed for each source with a certain probability that the binomial indicator is flipped to 0 or 1 (whichever number it is currently not). A probability p_1 is assigned to the first source and a probabilityp_2 is assigned to the second source. These probabilities rnage from 0, meaning no corruption, to .5, meaning complete corruption. For example, say the true value of y is 0, p_1 is 0, and p_2 is 0.5. Then the data that stems from the first source accurately represents the value of y which is 0. The data that stems from the second source will accurately represent the value of y in 50 percent of the samples, but the other 50 percent of the samples it flips to represent 1. This means that all of the information is coming from the first source while the second source tells us nothing about the true value of y.

In order to test how the classifiers perform with different level of corruption on the Double XOR data, we varied the probability of corruption for each XOR source in increments of 0.1 from 0 to 0.5. We performed this experiment three times, holding the dimension of each source static at three, five, and seven. The results of this experiment performed on five dimensional XOR are displayed in the table below. See the appendix for the results from the three dimensional and seven dimensional experiments.

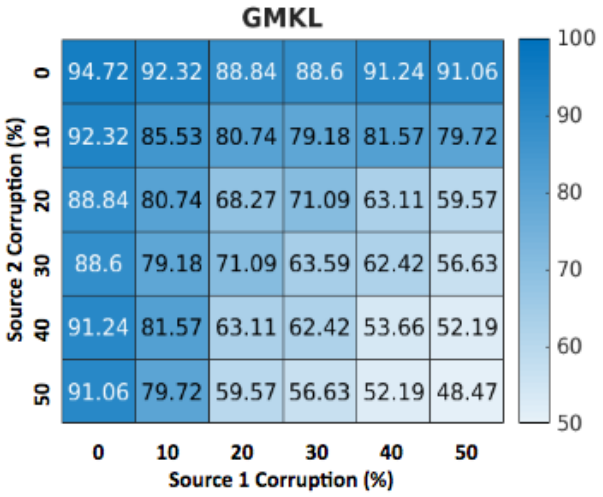


Figure 6: Put a caption here

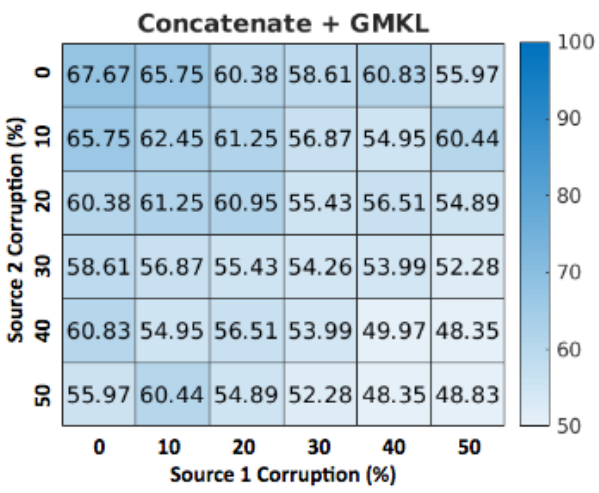


Figure 7: Put a caption here

We see that the results of Experiment 2 are consistent with the results of Experiment 1 in that GMKL consistenetly outperforms concatenated GMKL which consistenly outperforms the standard classifiers. We also see that the accuracy of GMKL is the least affected by one of the two sources becoming corrupted. This is evidence that GMKL effectively identifies which sources are useful and relies primarily on those sources.

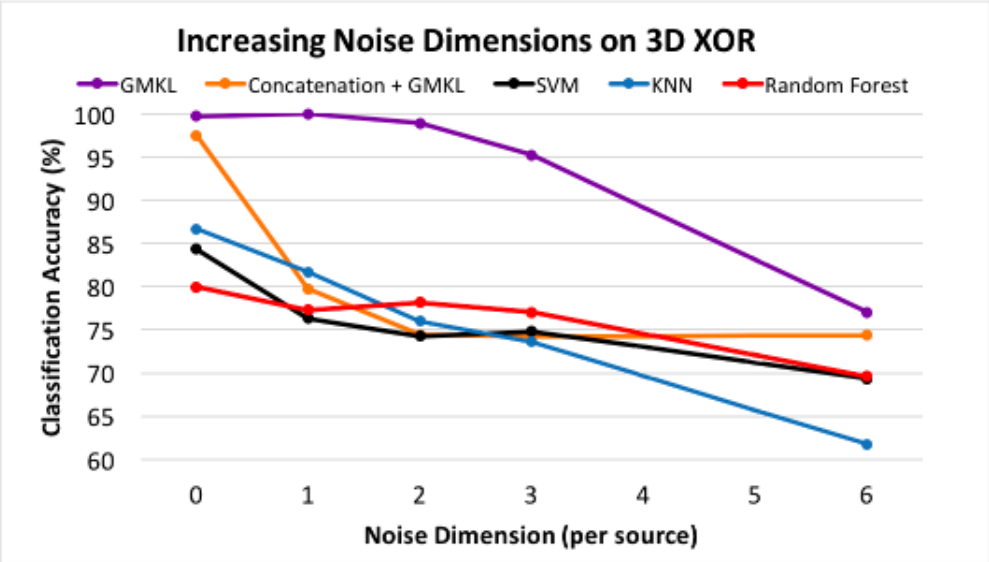
We see that the results of Experiment 2 are consistent with the results of Experiment 1 in that GMKL consistenetly outperforms concatenated GMKL which consistenly outperforms the standard classifiers. We also see that GMKL is the least affected by one of the two sources becoming corrupted. This is evidence that GMKL effectively identifies which sources are useful and relies primarily on those sources.

Experiment 3: Noise Dimensions

Another variable to explore is the number of noisy dimensions. This experiment explores how each classifier is able to handle additional dimensions that do not provide useful information regarding the class each point belongs to. Ideally, the classifiers are able to identify them as noise dimensions and ignores these dimension in their predictive model.

For this experiment, we added the same number of noise dimensions to each XOR source. We performed this experiment three times, when there were three, five, and seven XOR dimensions. We varied the proportion of dimensions that were noise dimensions. The proportion varied from no noise dimensions, to a quarter, third, half, and then fianlly two thirds noise dimensions. The purpose of measuring the proportion of noise variables as opposed to the number of noise variables is so that the three experiments over different XOR dimensions can be accurately compared.

The results from the three dimensional Double XOR experiment are represented in the plot below. The results from the five dimenional and seven dimensional XOR can be found in the appendix.



The purple line in the plot above shows how noise averse GMKL is while the accuracy of Concatenated GMKL falls when only a single noise dimension is added. When there are the same number of noise dimensions as there are useful dimensions, the accuracy of GMKL is still greater than 95 percent. This tells us that GMKL effectively ignores noise dimensions. It is worthy to note the Random Forest classifier is considered noise averse and is even less affected by noise than GMKL. However, GMKL is initially so much more accurate than Random Forest, that as far as we tested, the accuracy of GMKL remains superior to the Random Forest classifier.

Experiment 4: Corrupted Sine and XOR Sources

This experiment follows the same structure as Experiment 2 in terms of varying the corruption probabilities to each source. The difference in that instead of using two sources with the XOR structure, one XOR source was used and one Sine sources was used. We also varied the corruption probabilities in different increments. The corruption probability on each source ranged from 0 to 0.45 in increments of 0.15. We executed this experiment on three dimensional XOR and Sine data with a period of two. The results are displayed below.

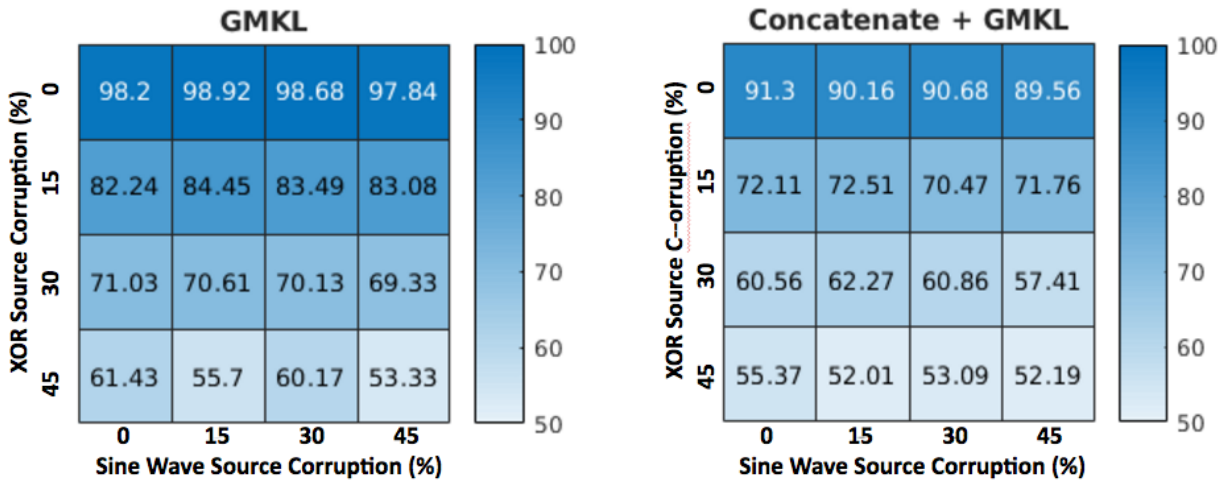


Figure 8: Put a caption here

Again we can see that GMKL outperforms Concatenated GMKL overall. From the heat maps above, we can observe that GMKL does not handle Sine data well. When the XOR source is heavily corrupted, the prediction accuracy of GMKL falls to barely better than a coin flip. On the opposite end of the heat map, we can note that GMKL performs very well when the Sine source is heavily corrupted as long as the XOR source remains intact. This means that GMKL is insensistive to unreliable data. GMKL is successfully able to identify the source that does not contribute to classification accuracy and ignore it.

Conclusion

From these experiment we have seen not only the usefulness of GMKL, but also with the issues that traditional classifiers face in data integration. Higher dimensional data, corrupted data sources, noisy dimensions, and variably useful data sources have all been shown to negatively effect the classification accuracy of traditional classifiers such as KNN, SVM and random forest. Not only does this illustrate the usefulness of GMKL as tool for classification and data integration, but the way in which GMKL generates kernels separately seems to indicate the important of treating your data sources as separate entities and the damage you will do to your classification accuracy if you concatenate it all together without any thought.

Future Work

For future work, first, we would like to see GMKL used on an real life data analysis problem too see if it shows the same utility that we saw previously. Second, since GMKL, and the creation of kernels is a fairly expensive operation to begin with, it would be nice to create a computationally less expensive version so that this could be run on larger datasets. Third, to show that GMKL is indeed up to the performance of other cutting edge algorithm, further comparison with other methods such as Neural Networks is necessary. And finally, if an investigation could be done of the theoretical properties of the GMKL and its relative performance compared to SVM would be appreciated to cement the useful GMKL as a data integration technique.