Multi-K Machine Learning Ensembles

Matthew Whitehead
Colorado College
Mathematics and Computer Science
14 E. Cache La Poudre St.
Colorado Springs, CO 80903
matthew.whitehead@coloradocollege.edu

Larry S. Yaeger
Indiana University
School of Informatics and Computing
919 E. 10th St.
Bloomington, IN 47408
larryy@indiana.edu

Abstract

Ensemble machine learning models often surpass single models in classification accuracy at the expense of higher computational requirements during training and execution. In this paper we present a novel ensemble algorithm called Multi-K which uses unsupervised clustering as a form of dataset preprocessing to create component models that lead to effective and efficient ensembles. We also present a modification of Multi-K that we call Multi-KX that incorporates a metalearner to help with ensemble classifications. We compare our algorithms to several existing algorithms in terms of classification accuracy and computational speed.

Introduction

Groups of machine learning models, called ensembles, can help increase classification accuracy over single models. The use of multiple component models allows each to specialize on a particular subset of the problem space, essentially becoming an expert on part of the problem. The component models are trained as separate, independent classifiers using different subsets of the original training dataset or using different learning algorithms or algorithm parameters. The components can then be combined to form an ensemble that has a higher overall classification accuracy than a comparably trained single model. Ensembles often increase classification accuracy, but do so at the cost of increasing computational requirements during the learning and classification stages. For many large-scale tasks, these costs can be prohibitive. To build better ensembles we must increase final classification accuracy or reduce the computational requirements while maintaining the same accuracy level.

In this paper, we discuss a novel ensemble algorithm called Multi-K that achieves a high-level of classification accuracy with a relatively small ensemble size and corresponding computational requirements. The Multi-K algorithm works by adding a training dataset preprocessing step that lets training subset selection produce effective ensembles. The preprocessing step involves repeatedly clustering the training dataset using the K-Means algorithm at different levels of granularity. The resulting clusters are then used as training datasets for individual component classifiers. The repeated clustering helps the component classifiers obtain different levels of classification specialization, ultimately leading to effective ensembles that rarely overfit. We also discuss a variation on the Multi-K algorithm called Multi-KX that includes a gating model in the final ensemble to help merge the component classifications in an effective way. This setup is similar to a mixture-of-experts system. Finally, we show the classification accuracy and computational efficiency of our algorithms on a variety of publicly available datasets. We also compare our algorithms with well-known existing ensemble algorithms to show that they are competitive.

Related Work

One simple existing ensemble algorithm is called bootstrap aggregating, or *bagging* (Breiman 1996). In bagging, component models are given different training subsets by randomly sampling the original, full training dataset. The random selection is done with replacement, so some data points can be repeated in a training subset. Random selection creates a modest diversity among the component models.

Bagging ensembles typically improve upon the classification accuracies of single models and have been shown to be quite accurate (Breiman 1996). Bagging ensembles usually require a large number of component models to achieve higher accuracies and these larger ensemble sizes lead to high computational costs.

The term *boosting* describes a whole family of ensemble algorithms (Schapire 2002), perhaps the most famous of which is called Adaboost (Domingo & Watanabe 2000), (Demiriz & Bennett 2001). Boosting algorithms do away with random training subset selection and instead have component models focus on those training data points that previously trained components had difficulty classifying. This makes each successive component classifier able to improve the final ensemble by helping to correct errors made by other components.

Boosting has been shown to create ensembles that have very high classification accuracies for certain datasets (Freund & Schapire 1997), but the algorithm can also lead to model overfitting, especially for noisy datasets (Jiang 2004).

Another form of random training subset selection is called *random subspace* (Ho 1998). This method includes all training data points in each training subset, but the included data point features are selected randomly with replacement. Adding in this kind of randomization allows components to

focus on certain features while ignoring others. Perhaps predictably, we found that random subspace performed better on datasets with a large number of features than on those with few features.

An ensemble algorithm called *mixture-of-experts* uses a gating model to combine component classifications to produce the ensemble's final result (Jacobs *et al.* 1991), (Nowlan & Hinton 1991). The gating model is an extra machine learning model that is trained using the outputs of the ensemble's component models. The gating model can help produce accurate classifications, but overfitting can also be a problem, especially with smaller datasets.

The work most similar to ours is the layered, cluster-based approach of (Rahman & Verma 2011). This work appears to have taken place concurrently with our earlier work in (Whitehead 2010). Both approaches use repeated clusterings to build component classifiers, but there are two key differences between the methods. First, Rahman and Verma use multiple clusterings with varying starting seed values at each level of the ensemble to create a greater level of training data overlap and classifier diversity. Our work focuses more on reducing ensemble size to improve computational efficiency, so we propose a single clustering per level. Second, Rahman and Verma combine component classifications using majority vote, but our Multi-KX algorithm extends this idea by placing a gating model outside of the ensemble's components. This gating model is able to learn how to weight the various components based on past performance, much like a mixture-of-experts ensemble.

Multi-K Algorithm

We propose a new ensemble algorithm that we call *Multi-K*, here formulated for binary classification tasks, but straightforwardly extensible to multidimensional classification tasks. Multi-K attempts to create small ensembles with low computational requirements that have a high classification accuracy.

To get accurate ensembles with fewer components, we employ a training dataset preprocessing step during ensemble creation. For preprocessing, we repeatedly cluster the training dataset using the K-Means algorithm with different values of K, the number of clusters being formed. We have found that this technique produces training subsets that are helpful in building components that have a good mix of generalization and specialization abilities.

During the preprocessing step the value for the number of clusters being formed, k, varies from K_{start} to K_{end}. K_{start} and K_{end} were fixed to two and eight for all our experiments as these values provided good results during pretesting. Each new value of k then yields a new clustering of the original training dataset. The reason that k is varied is to produce components with different levels of classification specialization ability.

With small values of k, the training data points form larger clusters. The subsequent components trained on those subsets typically have the ability to make general classifications well: they are less susceptible to overfitting, but are not experts on any particular region of the problem space. Figure 1 shows the limiting case of $k = 1$, for which a single classifier is trained on all of the training data. Figure 2 shows that as the value of k increases, classifiers are trained on smaller subsets of the original data.

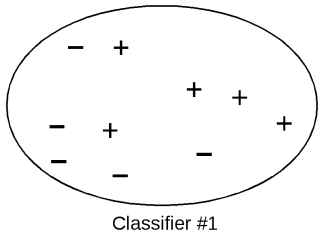

Figure 1: For $k = 1$, a single classifier is trained on the entire training set.

Larger values of k allow the formation of smaller, more specialized clusters. The components trained on these training subsets become highly specialized experts at classifying data points that lie nearby. These components can overfit when there are very few data points nearby, so it is important to choose a value for K_{end} that is not too large. This type of repeated clustering using varying values of k forms a pseudo-hierarchy of the training dataset.

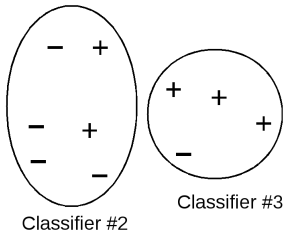

Figure 2: For $k = 2$, classifiers are trained on two disjoint subsets of the data, that overlap the $k = 1$ training set.

Figure 3 shows that as k is further increased, the clustered training datasets decrease in size allowing classifiers to become even more highly specialized. In this particular example, we see that classifier 6 will be trained on the same subset of training data as classifier 3 was above. In this way, tightly grouped training data points will be focused on since they may be more difficult to discriminate between. The clustering algorithm partitions the data in such a way as to foster effective specialization of the higher-k classifiers, thus maximizing those classifiers' ability to discriminate.

Following the clustering preprocessing step, each component model is trained using its assigned training data subset. When all the components are trained, then the ensemble is ready to make new classifications. For each new data point to classify, some components in the ensemble make classification contributions, but others do not. For a given new data point to classify, p, for each level of clustering, only those components with training data subset centroids nearest to p influence the final ensemble classification. Included component classifications are inversely weighted according to their centroid's distance from p.

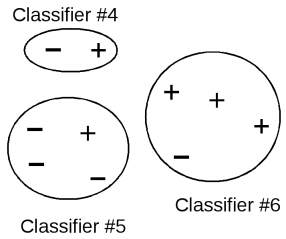

Figure 3: For $k = 3$, classifiers will be even more specialized because of their localized learning areas. If resulting clusters are identical, as is the case with clusters 3 and 6 in this example, only one cluster need be used to keep the final ensemble as small as possible.

Figure 4 shows an example where classifier #2 contributes to the final ensemble classification, since the data point to be classified (denoted by a '?') is nearest to its cluster centroid. Classifier #3 makes no contribution for this particular sample.

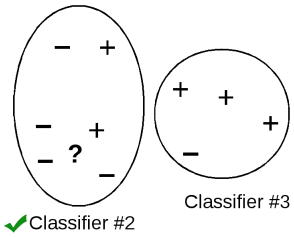

Figure 4: Classifier #2 contributes to the ensemble's final classification, but classifier #3 does not.

Figure 5 continues this example of component selection for the next level of clustering. At this level, classifier #5 is selected to contribute to the final classification and the other classifiers are ignored. Since classifier #5 was trained on a smaller training data subset than classifier #2 from Figure 4, it will be more specialized for the given problem. Its cluster centroid is also closer to the data point to be classified, so its classification weight will be greater.

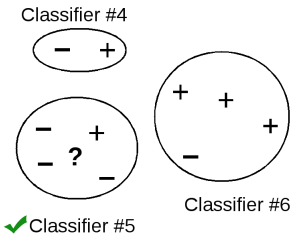

Figure 5: Classifier #5 is selected for ensemble contribution and is given a larger weight than classifier #2 from Figure 4.

The final classification is the weighted average of the n ensemble components' classifications in the current ensemble formation:

$$\frac{\sum_{i=0}^{n} w_i \cdot C_i(p)}{\sum_{i=0}^{n} w_i}$$

where $C_i(p)$ is classifier i's classification of target data point p and each w_i is an inverse distance of the form:

$$w_i = \frac{1}{dist(p, centroid_i)}$$

Pseudocode listing 1 shows the algorithm for clustering and training the component classifiers in Multi-K. Once the clustering and component classifier training is complete, then the ensemble is ready to classify new data points. Pseudocode listing 2 shows the algorithm for choosing the appropriate component classifiers given each new target data point to classify.

Given

D : training dataset
K_{start}: The number of clusters in the first level of clustering.
K_{end}: The number of clusters in the last level of clustering.

Ensemble Training Pseudocode

```
for k from K_start to K_end:
  cluster D into k clusters, D_ki, i ∈ [1, k]
  for i from 1 to k:
    train classifier C_ki on data D_ki
```

Pseudocode 1: Multi-K ensemble training.

Ensemble Formation Pseudocode

```
Given, p, a data point to classify

For each clustering (k):
  Find the cluster, D_ki, with centroid
  < D_ki > nearest to p
  Add C_ki, trained on D_ki, to ensemble
  Compute weight of C_ki with distance from
  < D_ki > to p
```

Pseudocode 2: Multi-K ensemble formation.

Finally, once the appropriate component classifiers have been selected, then the final ensemble classification can be calculated. Pseudocode listing 3 shows the algorithm for calculating the final classification based on a weighted sum of the outputs of the selected components.

Multi-KX Algorithm

The Multi-K algorithm used training dataset preprocessing to form effective component models. Final classifications were then performed by the entire ensemble by combining component classifications together based on the distance between $x_{predict}$ and the centroids of each of the component

Ensemble Classification Pseudocode

```
sum = 0
sum_weights = 0
for each classifier, C, in ensemble:
  weight_C = 1/dist(C,p)
  sum += weight_C * C's classification of p
  sum_weights += weight_C

final_classification = sum / sum_weights
```

Pseudocode 3: Multi-K ensemble classification.

training datasets. This technique works well, but we also thought that there may be non-linear interactions between component classifications and a higher accuracy could be gained by using a more complex method of combining components.

With this in mind, we propose a variation to Multi-K, called *Multi-KX*. Multi-KX is identical to Multi-K except in the way that component classifications are combined. Instead of using a simple distance-scaled weight for each component, Multi-KX uses a slightly more complex method that attempts to combine component outputs in intelligent ways. This intelligent combination method is achieved by the use of a gating metanetwork. This type of metanetwork is used in standard mixture-of-experts ensembles. This metanetwork's job is to learn to take component classifications and produce the best possible final ensemble classification. Figure 6 shows the basic setup of the ensemble.

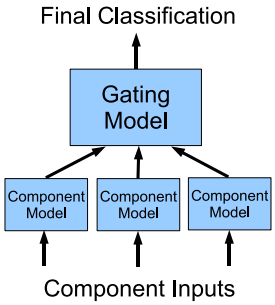

Figure 6: Ensemble classification using a gating model metanetwork.

The metanetwork can then learn the best way to combine the ensemble's components. This can be done by weighting certain components higher for certain types of new problems and ignoring or reducing weights for other components that are unhelpful for the current problem.

Building a Metapattern For the metanetwork to effectively combine component classifications, it must be trained using a labeled set of training data points. This labeled training set is similar to any other supervised learning problem: it maps complex inputs to a limited set of outputs. In this case, the metanetwork's training input patterns are made up of two different kinds of values. First, each classification value is included from all the ensemble's components. Then the distance between $x_{predict}$ and each cluster's centroid is also included. Figure 7 shows the general layout for a metapattern.

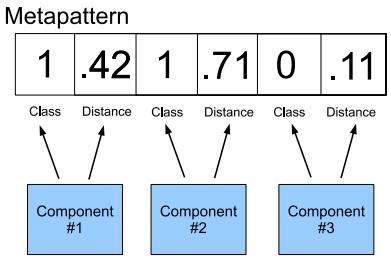

Figure 7: A Multi-KX metapattern. Inputs are grouped in pairs. The first number of each pair is the normalized component class prediction value (1 or 0 for this example). The second number is a measure of the distance between $x_{predict}$ and a component's training cluster centroid, normalized to the range (0, 1) where a value of 1 is used for the maximum distance centroid and a value of 0 is used for the minimum distance centroid. The metanetwork tries to learn the mapping from these values to the actual correct data point classification.

Experimental Results

To test the performance of our algorithms, we performed several experiments. First, we tested the classification accuracy of our algorithms against existing algorithms. Second, we measured the diversity of our ensembles compared to existing algorithms. Finally, we performed an accuracy vs. computational time test to see how each algorithm performs given a certain amount of computational time for ensemble setup and learning.

Accuracy Tests

Ensembles need to be accurate in order to be useful. We performed a number of tests to measure the classification accuracy of our proposed algorithms and we compared these results with other commonly-used ensemble techniques. We tested classification accuracy on a series of datasets from the UCI Machine Learning Repository (Asuncion & Newman 2007) along with a sentiment mining dataset from (Whitehead & Yaeger 2009). We performed a K-fold cross validation (with K=25) test using each algorithm on each dataset and we repeated each test ten times to ensure that the results were statistically stable. Each reported accuracy value is the mean of the resulting 250 test runs.

All accuracy tests were performed using support vector machines (Chang & Lin 2001) with linear kernels as the component classifiers, except we also compare our accuracies with boosting ensembles of decision stumps since the boosting algorithm is known to suffer less from overfitting with these component classifiers. For these tests, ensembles created with commonly used algorithms each had 50 component classifiers, as in (Breiman 1996).

Table 1 shows the classification accuracies for each tested algorithm and dataset. For each tested dataset, the most accurate result is shown in bold face. These results show that

the proposed algorithms are competitive with existing ensemble algorithms and are able to outperform all of those algorithms for some datasets. The telescope dataset in particular yielded encouraging results with more than a 3% increase in classification accuracy (a 16% reduction in error) obtained by the Multi-KX algorithm. Performance was also good on the ionosphere dataset with an almost 2% higher accuracy (a 17% reduction in error) than other ensemble algorithms.

The dataset that Multi-K performed most poorly on was the restaurant sentiment mining dataset, where it was more than 2% behind the subspace ensemble. Since that dataset uses an N-gram data representation model, the data is considerably more sparse than the other tested datasets. We hypothesize that the sparsity made the clustering and the resulting component classifiers less effective. None of the ensemble algorithms were able to outperform a single SVM on the income dataset. This again suggests that the nature of the dataset will occasionally determine which algorithms do well and which do poorly.

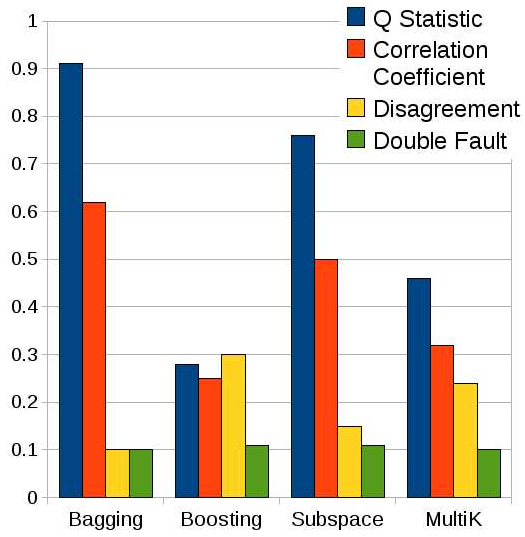

Figure 8: Measures of diversity for Multi-K ensembles on the heart dataset

Diversity Measurements

To form an effective ensemble, a certain amount of diversity among component classifiers is required. We measured the diversity of the ensembles formed by Multi-K using four different pairwise diversity metrics from (Kuncheva & Whitaker 2003):

- Q statistic - the odds ratio of correct classifications between the two classifiers scaled to the range -1 to 1.
- ρ - the correlation coefficient between two binary classifiers.
- Disagreement measure - proportion of the cases where the two classifiers disagree.
- Double-fault measure - proportion of the cases misclassified by both classifiers.

Figure 8 shows that the diversity of Multi-K ensembles generally falls in between algorithms that rely on random subsampling (bagging and subspace) and the one tested algorithm that particularly emphasizes diversity by focusing on previously misclassified training points (boosting). For example, values for the Q statistic and ρ were higher for the random methods and lower for boosting. The disagreement measure again shows Multi-K in the middle of the range.

Double fault values were nearly identical across all algorithms, suggesting that double fault rate is a poor metric for measuring the kind of diversity that is important to create ensembles with a good mix of generalization and specialization.

Combining Accuracy and Computational Efficiency

Since our main goal was to provide an algorithm that yielded high classification accuracies with the minimal amount of computational overhead, we performed a final combined accuracy and complexity test to show the relationship between the two for various ensemble algorithms. To do this, we ran existing ensemble algorithms with a wide variety of parameters that affect accuracy and training time. We then plotted a number of these training time/classification accuracy points, hoping to provide a simple, but informative way to compare the two across ensemble algorithms. We also ran each of the Multi-* algorithms and plotted each result as a single point on the graph because they have no free parameters to change. Each plot line is a best-logarithmic-fit for each existing algorithm to help see general trends. Figure 9 shows the results and has been normalized against the classification accuracy and computational time of a single SVM. Averaging across all tested datasets, Multi-K provided higher accuracy than other algorithms using anything less than about three times the compute time, and Multi-KX provided the highest accuracy of all, out to at least twice its computational costs.

Conclusions and Future Directions

We found that ensembles created using the Multi-* algorithms showed a good amount of diversity and had strong classification performance. We attribute this performance to a good mix of components with varying levels of generalization and specialization ability. Some components are effective over a large number of data points and thus exhibit the ability to generalize well. Other components are highly specialized at making classifications in a relatively small region of the problem space. The mix of both these kinds of components seems to work well when forming ensembles.

In the future, we hope to extend our method beyond binary to multi-class classification. In addition, we speculate that including additional characterizations of datasets and models as inputs to the gating network may further improve the accuracy of Multi-KX. We also are continuing work investigating alternative ways of preprocessing training datasets.

Ensemble	heart	bre	dia	iono	spam	tele	sent	inc
Single SVM	80.8	**96.7**	77.0	88.2	92.3	79.5	84.0	**85.2**
Bag	81.6	**96.7**	77.1	89.0	92.3	79.5	85.3	**85.2**
Boost SVM	81.6	96.6	77.1	88.9	**92.4**	79.6	84.0	84.5
Boost Stump	82.2	94.7	76.0	86.9	82.3	78.5	73.5	84.8
Subspace	**83.4**	96.7	76.4	88.2	91.7	77.8	**86.6**	84.6
Multi-K	83.1	96.5	77.2	88.6	92.1	79.8	84.3	85.1
Multi-KX	80.7	96.5	**77.4**	**90.9**	92.1	**82.8**	84.8	84.7

Table 1: Accuracy of algorithms with K-fold (K=25) tests - heart disease, breast cancer, diabetes, ionosphere, spam detection, telescope, restaurant sentiment mining, and income. For each tested dataset, the highest accuracy is shown in bold font.

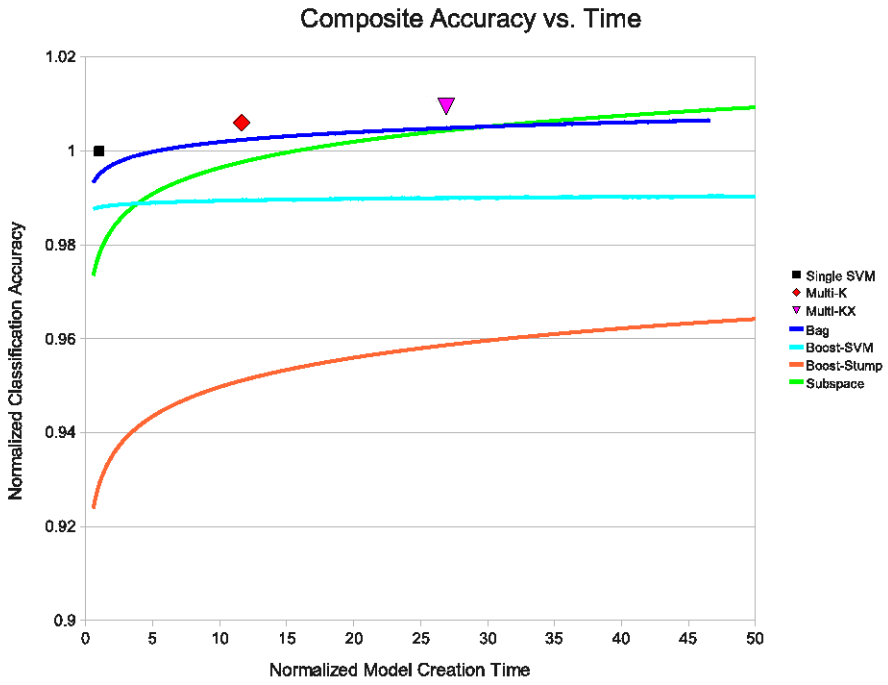

Figure 9: Normalized composite algorithm performance.

References

Asuncion, A., and Newman, D. 2007. UCI machine learning repository.

Breiman, L. 1996. Bagging predictors. *Machine Learning* 24(2):123–140.

Chang, C. C., and Lin, C. J. 2001. *LIBSVM: a library for support vector machines*. http://www.csie.ntu.edu.tw/∼cjlin/libsvm.

Demiriz, A., and Bennett, K. P. 2001. Linear programming boosting via column generation.

Domingo, and Watanabe. 2000. Madaboost: A modification of adaboost. In *COLT: Proceedings of the Workshop on Computational Learning Theory, Morgan Kaufmann Publishers.*

Freund, and Schapire. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *JCSS: Journal of Computer and System Sciences* 55.

Ho, T. K. 1998. The random subspace method for constructing decision forests. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* Volume 20(Issue 8):832–844.

Jacobs, R.; Jordan, M.; Nowlan, S. J.; and Hinton, G. E. 1991. Adaptive mixtures of local experts. *Neural Computation* 3:79–87.

Jiang, W. 2004. Boosting with noisy data: Some views from statistical theory. *Neural Computation* 16(4):789–810.

Kuncheva, L. I., and Whitaker, C. J. 2003. Measures of diversity in classifier ensembles. *Machine Learning* 51:181–207.

Nowlan, S. J., and Hinton, G. E. 1991. Evaluation of adap-

tive mixtures of competing experts. *Advances in Neural Information Processing Systems*.

Rahman, A., and Verma, B. 2011. Novel layered clustering-based approach for generating ensemble of classifiers. In *IEEE Transactions on Neural Networks*, 781–792. IEEE.

Schapire, R. E. 2002. The boosting approach to machine learning: An overview.

Whitehead, M., and Yaeger, L. 2009. Building a general purpose cross-domain sentiment mining model. In *Proceedings of the 2009 CSIE World Congress on Computer Science and Information Engineering*. IEEE Computer Society.

Whitehead, M. 2010. Creating fast and efficient machine learning ensembles through training dataset preprocessing. In *Ph.D. Dissertation, School of Informatics and Computing Indiana University*.