1 Scenario

Let us assume that we have a **small** labeled dataset from one domain (we are going to call this dataset the $TargetDataset$) and [usually] it is costly to obtain new labelled data from the same domain (so the quantity is limited).

Also, let us also assume that we have one or more **large** and labeled datasets from a ***related*** domains (we are going to call these datasets the $SourceDatasets$) and [usually] it is affordable to obtain new labelled data from these domains.

If we would like to build a model for the $TargetDataset$ using it alone, then the model will likely perform undesirably as the dataset is quite small!

The idea is to make maximum use of the $SourceDatasets$ to build a model for the $TargetDataset$ and classify data from its domain. In other words, we want to reuse data from the source tasks to augment the target task's training data (this is Instance Transfer Learning)

In this experiment, I have used the technique explained in a paper called *"Selective Transfer Between Learning Tasks Using Task-Based Boosting"* to gain knowledge from the *Source* data and use it in training a classifier for the $Target$ data. The proposed algorithm is called $TransferBoost$ and it is based on the classical *AdaBoost* Algorithm!

2 How TransferBoost Works (from the abovementioned paper)

As the source and target data are from different but related domains, the two tasks (i.e. source and target) have different distributions. Yet, some of the source tasks' data could have been drawn from the target task's distribution. Such data could then be used as additional training data for the target task.

$TransferBoost$ attempts to automatically select individual data from the source tasks to augment the target tasks training data. It automatically determines the weight to assign to each source instance in learning the target task's model, building on the *AdaBoost* algorithm. TransferBoost iteratively constructs an ensemble of classifiers, reweighting both the source and target data via two types of boosting: individual and task-based.
It increases the weight of individual mispredicted instances following AdaBoost. In parallel, it also performs task-based boosting by reweighting all instances from each source task based on their aggregate transfer to the target task.

In effect, TransferBoost increases the weight of source tasks that show positive transfer to the target task, and then reweights the instances within each task via AdaBoost.

3 Experiment I

3.1 The Data

- By looking at the table provided with the Eve data, I have extracted and merged labeled data from the Sapphire Channel (Sapphire Active/Inactive) for assays TS3 and TS6. This

is for strain *Plasmodium vivax*. This is going to be our $SourceDataset$

- I have also extracted labeled data from the Venus Channel (Venus Active/Inactive) for assay TS6. This is for strain *Plasmodium falciparum*. This is going to be our $TargetDataset$
- I have split the $TargetDataset$ into two subdatasets. One for training and one for testing. Notice that this training subdataset is going to be our actual $TargetDataset$
- Now our datasets look like:
  - $SourceDataset$ has 2781 instances (for Pv from TS3 and TS6 – file TS3-TS6-Pv-Source.arff)
  - $TargetDataset$ has 46 (for Pf from TS6 – file TS6-Pf-Target.arff)
  - $TestDataset$ has 1389 (for Pf from TS6 – file TS6-Test-Pf.arff)

3.2 Experimental Setup

The authors of the paper have provided the java source code of their implementation so I have downloaded it, plugged it into WEKA's source code and recompiled WEKA.
Remember that our $TargetDataset$ is quite small and our $SourceDataset$ is large and from a related domain.
We will use our $TestDataset$ for evaluation (it is from the same domain as $TargetDataset$)

Here is what I have done:

- I have built a classification model with the TransferBoost Algorithm using both the $Target$ and $Source$ Datasets to carry out Transfer Learning. I used *Decision Stump* as the base classifier.
- I have built classification models with WEKA's NaiveBayes, SVM, KNN and J48 Decision Trees using the $TargetDataset$ only. This is because usually we build models using data from the same domain!

3.3 Column Names:

I have abbreviated column names in the following tables to save display space. In order from left to right, they're as follows:
Percentage of Correct guesses,
Percentage of Incorrect guesses,
Area Under the Curve (for class Active),
Kappa statistic,
Mean Abs Error,
Root Mean Squared Error,
Relative Abs Error,
Root Relative Squared Error,
Precision (for class Active),
Recall (for class Active),
F-Measure (for class Active),
Error Rate

3.4 Experimental Stat Results 1

In this experiment I did 10 fold cross validation using the target dataset only (built models using the target dataset only and used it for the validation)

	corr	incorr	auc	kap	mae	rmse	rae	rrse	prec	rec	fm	err rate
TL	95.65%	4.34%	0.97	0.72	0.04	0.21	23.81%	67.52%	1	0.6	0.74	0.04
NB	97.82%	2.17%	1	0.89	0.02	0.14	10.39%	46.77%	0.83	1	0.9	0.02
J48	91.3%	8.69%	0.86	0.61	0.08	0.29	41.56%	93.55%	0.57	0.8	0.66	0.08
SMO	93.47%	6.52%	0.7	0.54	0.06	0.25	31.17%	81.02%	1	0.4	0.57	0.06
IBk	91.3%	8.69%	1	0.3	0.07	0.21	35.22%	67.32%	1	0.2	0.33	0.08

3.5 Experimental Stat Results 2

In this experiment I did 10 fold cross validation using the test dataset (built models using the target dataset and used the test dataset for the validation)

	corr	incorr	auc	kap	mae	rmse	rae	rrse	prec	rec	fm	err rate
TL	99.56%	0.43%	0.99	0.86	0	0.05	12.53%	42.36%	0.86	0.86	0.86	0
NB	97.55%	2.44%	0.99	0.55	0.02	0.15	76.65%	125.25	0.39	1	0.56	0.02
J48	99.49%	0.5%	0.89	0.83	0	0.06	19.11%	55.34%	0.85	0.81	0.83	0
SMO	99.64%	0.35%	0.9	0.87	0	0.05	11.27%	48.05%	0.94	0.81	0.87	0
IBk	99.42%	0.57%	0.97	0.78	0	0.06	18.53%	49.37%	0.93	0.68	0.78	0

3.6 Experimental Stat Results 3

In this experiment I did validation using the test dataset (built models using the target dataset and used the test dataset for the validation)

	corr	incorr	auc	kap	mae	rmse	rae	rrse	prec	rec	fm	err rate
TL	99.56%	0.43	0.99	0.87	0	0.06	3.35%	38.34%	0.8	0.95	0.87	0
NB	91.72%	8.27%	0.95	0.25	0.08	0.28	60.47%	173.46%	0.16	1	0.27	0.08
J48	98.2%	1.79%	0.99	0.62	0.01	0.13	13.14%	80.89%	0.46	1	0.63	0.01
SMO	99.28%	0.71%	0.97	0.8	0	0.08	5.25%	51.16%	0.7	0.95	0.8	0
IBk	99.35%	0.64%	0.95	0.73	0.01	0.07	9.61%	44.37%	1	0.59	0.74	0

3.7 Experimental Results

After building the models as explained above, I have evaluated them using the $TestDataset$ which is of the same domain as the $TargetDataset$. I have counted Actual vs Predicted results. The following table shows how many miss-classifications each model makes:

TransferBoost	NaiveBayes	SVM	KNN	J48 Decision Trees
6	115	10	9	25

Observe that the TransferBoost model (the one that does Transfer Learning) makes less classification errors meaning it outperforms models built using the $TargetDataset$ alone.

4 Experiment II

In this experiment, I am going to try and do TL at assay level. Meaning I will use Active/Inactive labelled datasets from the eve data (assays TS3,4,5,6,7)

4.1 The Data

- For the *SourceDataset*, I have used TS3 (1346 instances (4 Active) – file TS3-Labeled.arff)
- I have randomly split TS5 into two datasets:
  - *TargetDataset* ... 278 instances (3 Active) – file TS5-Labeled-Target.arff
  - *TestDataset* ... 1116 instances (5 Active) – file TS5-Labeled-Test.arff

4.2 Experimental Setup

Exactly the same as Experiment I

4.3 Experimental Stat Results 1

In this experiment I did 10 fold cross validation using the target dataset only (built models using the target dataset only and used it for the validation)

	corr	incorr	auc	kap	mae	rmse	rae	rrse	prec	rec	fm	err rate
TL	99.64%	0.35%	0.68	0.79	0	0.05	14.34%	57.83%	1	0.66	0.8	0
NB	98.2%	1.79%	0.67	0.43	0.01	0.13	76.73%	130.29%	0.33	0.66	0.44	0.01
J48	98.56%	1.43%	0.66	0.32	0.01	0.12	61.86%	117.27%	0.33	0.33	0.33	0.01
SMO	98.56%	1.43%	0.66	0.32	0.01	0.11	57.1%	115.67%	0.33	0.33	0.33	0.01
IBk	98.92%	1.07%	0.86	0	0.01	0.09	48.77%	89.55%	0	0	0	0.01

4.4 Experimental Stat Results 2

In this experiment I did 10 fold cross validation using the test dataset (built models using the target dataset and used the test dataset for the validation)

	corr	incorr	auc	kap	mae	rmse	rae	rrse	prec	rec	fm	err rate
TL	99.73%	0.26%	0.99	0.66	0	0.04	26.09%	74.67%	0.75	0.6	0.66	0
NB	98.65%	1.34%	0.99	0.39	0.01	0.11	135.78%	173.48%	0.25	1	0.4	0.01
J48	99.82%	0.17%	0.89	0.79	0	0.04	18.1%	63.34%	0.8	0.8	0.8	0
SMO	99.73%	0.26%	0.79	0.66	0	0.05	27.15%	77.58%	0.75	0.6	0.66	0
IBk	99.55%	0.44%	1	0	0	0.04	32.77%	61.41%	0	0	0	0

4.5 Experimental Stat Results 3

In this experiment I did validation using the test dataset (built models using the target dataset and used the test dataset for the validation)

	corr	incorr	auc	kap	mae	rmse	rae	rrse	prec	rec	fm	err rate
TL	99.73%	0.26%	0.99	0.57	0	0.05	14.43%	76.81%	1	0.4	0.57	0
NB	99.28%	0.71%	0.79	0.49	0	0.08	38.46%	125.43%	0.36	0.8	0.5	0
J48	99.55%	0.44%	0.69	0.44	0	0.06	24.03%	91.42%	0.5	0.4	0.44	0
SMO	99.73%	0.26%	0.7	0.57	0	0.05	14.42%	76.81%	1	0.4	0.57	0
IBk	99.55%	0.44%	0.99	0	0	0.04	23.05%	73.11%	0	0	0	0

4.6 Experimental Results

After building the models as explained above, I have evaluated them using the $TestDataset$ which is of the same domain as the $TargetDataset$. I have counted Actual vs Predicted results. The following table shows how many miss-classifications each model makes:

TransferBoost	NaiveBayes	SVM	KNN	J48 Decision Trees
3	8	3	5	5

Observe that the TransferBoost model (the one that does Transfer Learning) makes less classification errors meaning it outperforms models built using the $TargetDataset$ alone.