Solar Flare Data: Semi-supervised Learning and Transfer Learning

Amulya Kallakuri (kallakur@usc.edu)
Nithyashree Manohar (nithyash@usc.edu)

December 5, 2022

Project Type 1: System based on real-world data

1 Abstract

Solar flares pose a risk to satellites and telecommunication infrastructure, several space-based solar observatories observe solar flares for this reason. The solar flare data set from the KEEL repository is a classification problem and aims to predict the class of the solar flare. There are two versions of this dataset, one with 40% labeled and 60% unlabeled data which was used for semi-supervised learning, and one with 100% labeled data that was used for transfer learning.
For the semi-supervised learning task, we implemented Self-Training models, Label Propagation algorithm, Label Spreading algorithm, QN-S3VM algorithm and the Random Walk algorithm. From these, it was found that QN-S3VM is the best performing algorithm with an accuracy of 87% on the test data.
For the transfer learning task, we implemented TrAdaBoost and Subspace Alignment. Further models were assessed in both of these and compared with their results analyzed. From these, it was observed that the TrAdaBoost algorithm yielded higher scores and the Subspace Alignment algorithm, lower.

2 Introduction

2.1 Problem Type, Statement and Goals

The sun is now in its active phase of the 11-year solar cycle, and the intensity of the solar flares is expected to increase. Being able to predict the class of the solar flares will help to minimize the damage caused by these flares. With a rise in the number of satellites and a significant improvement in the telecommunication world, a severe solar storm could lead to a global blackout. This is why being able to predict these flares is an important problem. This is a classification problem that predicts the class of the solar flares. In this project, we perform semi-supervised learning and transfer learning tasks.

For the semi-supervised learning task, the goal is to use 40% labeled and 60% unlabeled data to predict the class a solar flare belongs to. The goal is to apply the algorithms taught in EE660 to this problem to achieve a good test accuracy score.

This problem is an apt example of the incredible results transfer learning will and is capable of achieving. Transfer learning is valuable in this case since it is not always that we will have all the data labeled and in a comfortable format for us to work with. Often, in a real world problem, we do not have sufficient enough information to learn and predict with the help of simple classification models. As is shown in this project, we will use transfer learning on a source dataset and apply it to the target dataset.

2.2 Literature Review

Paper [6] was extremely useful in understanding TrAdaBoost and Boosting for Transfer Learning.
Paper [11] was useful for understanding why the error rate for TrAdaBoost had to be lesser than 0.5

2.3 Our Prior and Related Work

- Nithyashree Manohar: I do not have experience working with semi-supervised learning algorithms prior to EE660. However, I have learned and worked with several underlying algorithms SSL uses. The self learning algorithms were implemented using Logistic Regression and Support Vector Machines, S3VM is an extension of Support Vector Machines for semi-supervised learning. Logistic Regression, Support Vector Machines and the Expectation Maximization algorithms are few topics I have worked with previously.

- Amulya Kallakuri: I have had a fair idea of what transfer learning entails due to my research on Zero-shot Learning and One-shot Learning, along with learning from dictionary embeddings with non-linear kernels. However, I did not have a sufficient enough theoretical understanding of this prior to my taking this class. This project has significantly increased my understanding of transfer learning in the theoretical as well as practical domain.
  The both of us are amazed by the capabilities of semi-supervised learning and transfer learning that can be applied to this problem. Both SSL and TL are incredible topics to work with and we are truly grateful for having taken this course and applied it to a problem of such significance.

2.4 Overview of Our Approach

Labeled data is often rare to find in real-world applications. Labeling data can be time-consuming and expensive. Therefore, extension semi-supervised learning uses both labeled and and unlabeled data to build models and extension transfer learning learns from one set of data and implements its learning on another set of data.

2.4.1 Semi-Supervised Learning

The semi-supervised learning task was implemented using the algorithms listed below and compared using metrics such as accuracy score, f1 score, precision, recall and support. Baseline systems, Logistic Regression and Support Vector Machines were implemented to compare the performance with the semi-supervised learning models.

- Label Propagation: This is a graph based technique that creates a graph that connects the data points in the training data set and propagates known labels through the edges of the graph to label unlabeled data points. It computes a similarity matrix between samples and uses a KNN-based approach to propagate samples. For this task, we used the sklearn package to implement label propagation.

- Self-Learning Algorithm: In a self training model, we train a classifier to predict labels for labeled data. We then use this trained classier to predict pseudo-labels for the unlabeled data. Concatenate the labeled and unlabeled data to train the classifier again and use it to predict the labels for the test data. For this task, we trained the classifier using Logistic Regression and Support Vector Machines.

- QN-S3VM: The goal of a standard support vector machine is to find a hyperplane which separates classes well such that the margin is maximized. The Quasi-Newton Optimization for Semi-Supervised Support Vector Machines takes unlabeled patterns into account by searching for a partition such that a subsequent application of a (modified) support vector machine leads to the best result [7]. For this task, we implemented a one-vs-rest approach to each of the six classes.

- Label Spreading: This algorithm works similar to label propagation but adds a regularization term to be more robust to noise. For this task, we used the sklearn package to implement label spreading.

- Random Walk: This algorithm works by taking into account the entire unlabeled network and the information provided by it. Few labeled instances are inserted into the network by a specific weight composition responsible for creating a bias to the classification process [5]. This bias is taken into account when the limiting probabilities are calculated and intrinsic relations among labeled and unlabeled vertices in a network are established.

2.4.2 Transfer Learning

The transfer learning task was implemented in two main categories: TrAdaBoost and Subspace Alignment. They were both compared with their accuracy and error scores. The baseline systems were SVM, SVMt and AUX.

- TrAdaBoost:
  - Custom Implementation
    - Decision Tree
    - Linear SVC
    - SVC
    - Baseline AdaBoost from SKLearn
  - Python Adapt Package Implementation
    - Linear Discriminant Analysis
    - Linear SVC
    - MLP Classifier
    - Gaussian Process Classifier
- Subspace Alignment (both custom and existing Adapt):
  - MLP Classifier
  - K Neighbors
  - Linear SVC
  - SVC
  - Gaussian Process
  - Decision Tree
  - Random Forest
  - AdaBoost
  - Linear Discriminant Analysis

3 Implementation for Semi-Supervised Learning

3.1 Dataset

In this data set, the target variable is a multi-class categorical attribute and denotes the class of solar flare it belongs to. There are 11 input features and 2465 data points divided into labeled (1782 data points), unlabeled (576 data points) and test data (107 data points). There is no missing data.

Name of the column	Type	Description
LargestSpotSize	Categorical	Flares are divided into 6 types based on size
SpotDistribution	Categorical	Flares are divided into 3 types based on distribution
Activity	Categorical	Flares are divided into 2 types based on activity - reduced and unchanged
Evolution	Categorical	Flares are divided into 3 classes based on evolution - decay, no growth and growth
Prev24Hour	Categorical	Measures activity in the last 24 hours and divides it into 3 types - $\leq moderateflare, moderate, \geq moderateflare$
HistComplex	Categorical	Flares are divided into 2 classes based on their history
BecomeHist	Categorical	Flares are divided into 2 classes based on historically complex on this pass across the sun's disk
Area	Categorical	Denotes the area of the flare - small and large
C-class	Integer	Number of common flares produced in the last 24 hours
M-class	Integer	Number of moderate flares produced in the last 24 hours
X-class	Integer	Number of severe flares produced in the last 24 hours
Class	Categorical	Target variable - 6 classes of flares

3.2 Dataset Methodology

The data set from the KEEL repository has three files - labeled data, unlabeled data and test data containing 959, 959 and 107 data points respectively. However, the unlabeled data set has a mix of both labeled and unlabeled data points. We performed operations to separate the unlabeled data from the labeled data. After this operation, the labeled data set has 959 data points, unlabeled data set has 576 data points and the test data set has 107 data points. The test data set was used after model training for testing purposes.

3.3 Preprocessing, Feature Extraction, Dimensionality Adjustment

Out of the 11 input features, there are 8 categorical features and 3 integer features. Out of the 8 categorical features, Activity, HistComplex, BecomeHist, and Area have only two classes. We converted these four features to binary values, 1 for yes and 0 for no. We generated dummies for the other four input features - LargestSpotSize, SpotDistribution, Evolution, and Prev24Hour. We performed min max scaling on the integer input features - C-class, M-class and, X-class. The range of C-class is 0-8, the range of M-class is 0-5 and the range of X-class is 0-2. Min-max scaling is similar to z-score normalization in that it will replace every value in a column with a new value using a formula. In this case, that formula is:

$$\text{m} = (\text{x} - \text{xmin}) \ / \ (\text{xmax} - \text{xmin})$$

where, m is our new value, x is the original cell value, xmin is the minimum value of the column, and xmax is the maximum value of the column. [10] The labeled data set in unbalanced and each class contains varying number of data points.

Class	Number of data points
0	133
1	189
2	216
3	85
4	39
5	297

Number of datapoints in each class

To address this issue we used Synthetic Minority Oversampling Technique. SMOTE is a process by which the minority class is over-sampled to create equal number of data points in each class. This is done by identifying the minority class and using K-Nearest Neighbors algorithm to generate synthetic data. After performing this technique, each class has 297 data points.

3.4 Training Process

3.4.1 Self-Training Algorithm

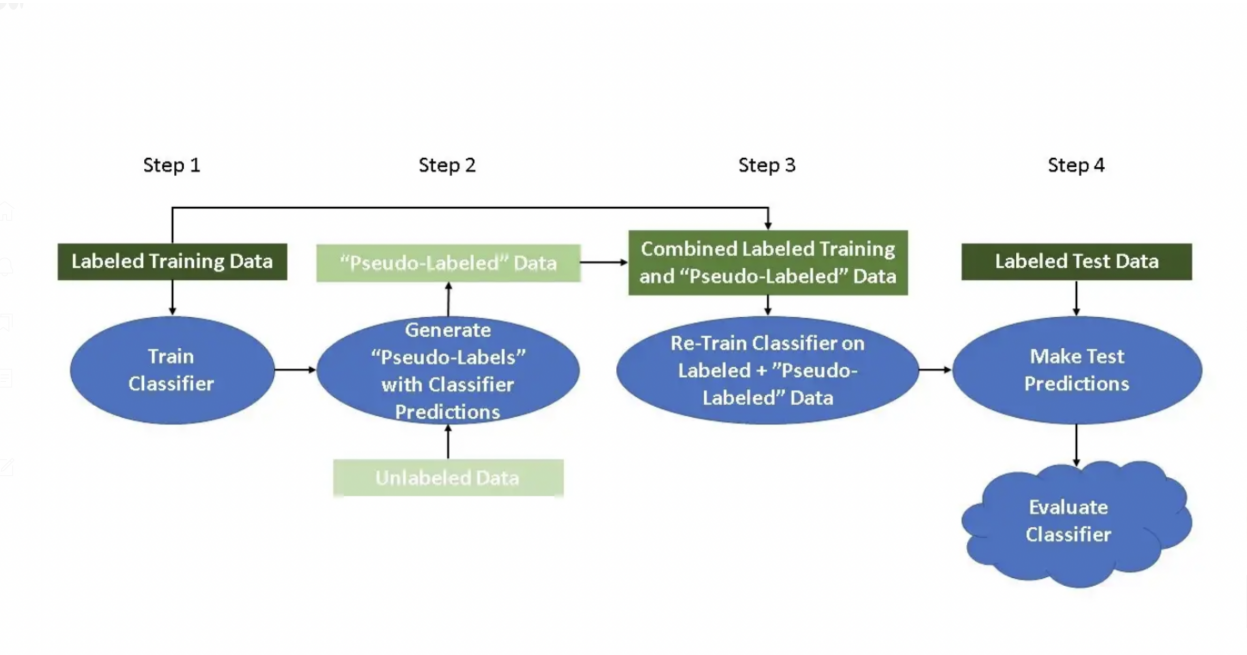

Figure 1: Self-Training

- Algorithm
  - The labeled data set is divided into train and test set. Then, the train set is used to train a classifier.
  - The trained classifier is used to make predictions on the unlabeled data. The predictions with the highest probability from the classifier are taken as pseudo-labels.
  - The labeled train data and the pseudo-labeled data are concatenated and the classifier is trained again.
  - This classifier is now used to make predictions on the test data.
- Justification: The self-training algorithm was chosen since the classifier can be trained on any algorithm of choice and the performance can be compared. 1-Nearest Neighbor was an algorithm taught in class but the time and space complexity of 1-NN makes it hard to implement. Hence, we implemented self-training using Logistic Regression and Support Vector Machines.
- Parameters
  - The self-training part of the Logistic Regression algorithm was implemented from scratch with the parameter **max_iter = 100**. This parameter was chosen to let the classifier have enough time to reach convergence.

- Support Vector Machines were implemented using sklearn self-training package with the following parameters **kernel='rbf', probability=True, C=1.0, gamma='scale', random_state=0**. These parameters were chosen through a trial and error process.

- Complexity
  - Logistic Regression Train Time Complexity=O(n*m), Test Time Complexity=O(m), Space Complexity = O(m)
  - Support Vector Machines Train Time Complexity=$O(n^2)$, Test Time Complexity=O(n'*m), Space Complexity = O(n*m)
    where, n = number of training examples, m = number of features, n' = number of support vectors.

- Results: The Logistic Regression algorithm has a test accuracy of 75% and the Support Vector Machine algorithm has a test accuracy of 77%.

3.4.2 Label Propagation and Label Spreading Algorithm

- Algorithm
  - Each of the training data points have their corresponding labels. A network is created using these data points and their labels.
  - These labels propagate through the graph. At every iteration, each node updates its label to the one that the maximum numbers of its neighbours belongs to.
  - This algorithm reaches convergence when each node has the majority label of its neighbours. It stops if either convergence, or the user-defined maximum number of iterations is achieved.

  Label Spreading and Label Propagation work in a similar way expect, Label Spreading algorithm uses symmetric normalized graph Laplacian matrix in its calculations, while Label Propagation employs a random walk normalized Laplacian. Label Propagation uses hard clamping, which means that the labels of the originally labeled points never change. Meanwhile, Label Spreading adopts soft clamping controlled through a hyper parameter, alpha, which specifies the relative amount of information the point obtains from its neighbors vs. its initial label information. [4]

- Justification: Label Spreading and Label Propagation are graph based algorithms that work well when we have only a small number of labeled examples and want to apply auto-labeling on a large amount of unlabeled data. These algorithms have an excellent run time and simple algorithm.

- Parameters
  - These algorithms were implemented using the sklearn package and the parameters were chosen through a trial and error process. The parameters for the both the algorithms are as follows, Label Propagation - **n_neighbors = 5, kernel='rbf'**. Label Spreading - **kernel='rbf', gamma=20, n_neighbors=7, alpha=0.5, max_iter=100, tol=0.001, n_jobs=-1,**

- Results: Both the algorithms yield a test accuracy of 76%

3.4.3 QN-S3VM Algorithm

- Algorithm
  To implement QN-S3VM, we used the one-vs-rest approach and converted the multi-class problem into a two class problem. The QNS3VM algorithm is a type of Transductive SVM with the quasi-newton optimization method. The quasi-newton method is an alternative way of finding the local minima of the loss function. This method first obtains an initial candidate class in a supervised manner and subsequently, makes use of increasing proportions of unlabeled data instances to find new insights. Different input features (Unigram, Bigram, TF-IDF, and POS) and ratios of labeled data are explored. No different base estimators are required since the method runs on Support Vector Machines only. [9]

- Justification: This algorithm typically has a local minima and the weight values can depend on the starting values. But, even with this issue it tends to work well. S3VM tends to work well if unlabeled data form clusters that have some separation that are consistent with labeled data. We believe that the seperation of clusters is consistent with the labeled data since this is one of our best performing algorithms.
- Complexity: The run-time complexity of QN-S3VM is O(n^3)
- Results: This algorithm yields an accuracy of 87% for all classes

3.4.4 Random Walk

- Algorithm
  A graph with labeled and unlabeled data points is generated. If we encounter an unlabeled node, i, in our graph, then, we randomly walk to neighboring nodes. Our choice of edges will be determined by their weight, so we are more likely to take edges with higher weight.
  When we encounter a labeled node, we will stay there and stop the walk. For each of the labeled nodes, the probability of ending at that particular node is given by the following equations.
  This random walk will give rise to a probability distribution over all the labeled nodes. Let's imagine that as a vector, $p\_i \in R^{N\_labeled}$. Then p_ij indicates the probability that we start at unlabeled node, i, and end at the labeled node, j.
  Repeating these steps for each unlabeled node and stacking those vectors in a matrix results in a matrix, $P \in R^{N\_unlabeled \times N\_labeled}$.
  In case of semi-supervised learning, to make a prediction on any of the unlabeled nodes, we can use the infinity matrix as a linear smoother. So our prediction is $y_{pred} = P/y\_labeled$
- Justification: The main advantage of this algorithm is that it is simple to use and can be easily handled flows around complicated boundaries.
- Results: Random Walk algorithm yields an accuracy of 79.4%

3.5 Model Selection and Comparison of Results

4 Final Results and Interpretation for Semi-Supervised Learning

4.1 Final results

SL Classifier	Train Accuracy	Test Accuracy
Logistic Regression	78%	75%
Support Vector Machines	77%	75%

Accuracy scores for Supervised Learning Algorithms

SSL Classifier	Train Accuracy	Test Accuracy
Self-Training Logistic Regression	78%	75%
Self-Training Support Vector Machines	78%	77%
Label Propagation	84%	76%
Label Spreading	84%	76%
QN-S3VM	89%	87%
Random Walk	82%	79%

Accuracy scores for Semi-Supervised Learning Algorithms

4.2 Interpretation

From the results it can be seen that most of the semi-supervised models perform better than the supervised learning models.
However, the accuracy of the self-training Logistic Regression algorithm is the same as that of the supervised learning methods. This algorithm runs only for three iterations adding 80 unlabeled samples to the labeled data set. The predictions do not have a high probability, the threshold is set at 90% and hence this algorithm does not perform too well.
The self-training Support Vector Machine algorithm adds 475 unlabeled samples to labeled instances and runs for 8 iterations performing slightly better than the supervised classifier and the self-training Logistic Regression algorithm. Since this algorithm runs more iterations our assumption is that this classifier outputs labels with a higher probability and hence, it performs better.
Label Propagation and Label Spreading results in the same performance yielding an accuracy of 76%. In this case, our assumption is that the transition matrix constructed by these algorithms are similar and hence they yield the similar results.
QN-S3VM, uses the quasi-newton optimization method to train the classifier. This is the best performing algorithm and the reason for this could be the one-vs-rest approach. The multi-class problem is converted into a two class problem which could possibly explain the increase in performance.
Random walk yields an accuracy of 79% which is better than most of the other classifiers. This is a graph-based algorithm that uses weights to update labels.

5 Implementation for Transfer Learning

5.1 Dataset

The dataset for the transfer learning task is the same as the semi-supervised learning task except that the number of datapoints in this task is a total of 1066. Refer to [1].

5.2 Dataset Methodology

The KEEL repository has a completely labeled dataset of size 1066 that we shall use for the transfer learning task of this project. We performed K Means clustering on this dataset with two clusters to separate it into source and target data based on cluster number. We used the points from class 0 to be the target data, and class 1 to be the source data. This target data was then split in two and chosen as target and target test data. This is the same data used in TrAdaBoost as well as Subspace Alignment for all models.

5.3 Preprocessing, Feature Extraction, Dimensionality Adjustment

- Modified the original .dat file by excluding the header lines and converting to a csv file using pandas.read_csv.
- Converted the categorical columns to numeric. Dropped them from the original dataframe (let's call this *df*), used *LabelEncoder* to convert to integers, and then inserted them into *df* using the same column number.
- Dropped the label (y) column from *df* and assigned to a separate variable named *classes*.
- Converted the SL dataset into a TL dataset by performing *K-Means* clustering. Class 1 from the predicted y after *K-Means* was considered *source (xs)* data and class 0 as *target (xt)* data.
- *xt* was split in two halves with one half considered actual *target (xt)* and the other half *test (xtest)* data.

5.4 Training Process:

5.4.1 Tranfer Learning with TrAdaBoost:

- Apart from the SSL, transfer learning (TL) was performed on the completely labeled version of the dataset used in the SSL problem.
  For this, the following approach was followed:
  - Pending the preprocessing, data extraction and the dimensionality adjustment, the following approaches were followed:
    * The baseline was considered AdaBoost from SKLearn with number of estimators as 3. This was then fit and predicted on *Xs* and *Xt* to give accuracy scores. The scores are as mentioned in Table 1. Along with this, SVM, SVMt and AUX models were also used.
    * Thanks to Zhiruo Zhou, [12] and [8], we had an idea of where to start for the TrAdaBoost implementation in Python.
    * A TrAdaBoost class was implemeted which took in the number of times it should run N, and a base estimator. This class had the functions: *_calculate_weights*, *_calculate_error_rate*, *fit*, *_predict_one* and *predict*.
      · *_calculate_weights* (argument *weights*): raveled, summed and returned as an array of *weights / sum of weights*
      · *_calculate_error_rate* (arguments *y_true, y_pred, weights*): same as above except returned the sum of *weights / sum of weights* multiplied by the absolute value of *(y_true – y_pred)*
      · *fit* (arguments *source, target, source_label, target_label*): fit the source and target data with the additional condition that the error rate must not equal 0 or be greater than 0.5 according to [1]. This would then run N number of times calculating weights, cloning the estimator N times and then predicting on the estimator in that iteration.
      · *_predict_one* (argument *x*): takes in argument in the form of an array-like target label of a single instance from each iteration and returns 0 or 1 depending on the condition.
      · *predict* (argument *x_test*): predicts *x_test* with the estimator provided above, maps *_predict_one* to each of the items in this list and returns the predictions.
  - The metric used in the custom implementation of TrAdaBoost was the ROC-AUC Score from Sci-kit Learn.
- Apart from this implementation of TL with TrAdaBoost, the Adapt package [3] was also implemented. In this method, SVM, SVMt, AUX (Balanced Weighting) and TrAdaBoost were implemented and compared.
- This would make a total of two main methods implemented for TrAdaBoost, the first for which four models were compared, and the second with four more methods.
- Justification: This method was chosen since this is the most apt method to perform transfer learning, as substantiated by the lecture as well.
- Parameters: The parameters were chosen based on sufficient experimentation with various values and based on the performance of the method with these values. These were corroborated with heuristics.

5.4.2 Tranfer Learning with Subspace Alignment (SA):

- The initial dataset preprocessing was done as per the method used in the Transfer Learning with TrAdaBoost.
- Pending the preprocessing, feature extraction and dimensionality adjustment, the following was implemeneted:
  - The paper [6] was instrumental in our understanding of the subspace aligment.

  - Thanks to our Homework 4 for EE660, we were able to gain decent knowledge of how to go about implementing subspace alignment.
  - A subspace alignment class was created which comprised of the following functions: *fit, transform* and *score* which would fit, transform and score the data as would be done in a machine learning model class.
  - Three other additional methods *get_clf, experiment* and *execute* were added which would get the classifier based on a string, would carry out the experiment and would execute it respectively.

- The implementation of Subspace Alignment from the Python Adapt package was also done to compare the results and note their core differences. [2]
- Justification: This method along with the TrAdaBoost algorithm, are the two methods with most research done in the field of transfer learning with sufficient background for implementation in Python.
- Parameters: The parameters were chosen as given in the lecture and by recommended values. This was also chosen with heuristics and by trail-and-error of which parameters give the most significant values.

6 Final Results and Interpretation for Transfer Learning

6.1 Transfer Learning with TrAdaBoost:

Base Classifier	Type of Accuracy	Accuracy Score
Decision Tree	Train	0.2352
	Target	0.0902
	Target Test	0.1041
Linear SVC	Train	0.2352
	Target	0.0902
	Target Test	0.1041
SVC	Train	0.2352
	Target	0.0902
	Target Test	0.1041
Baseline AdaBoost from SKLearn	Train	0.8714
	Target	0.4722
	Target Test	0.5069

Scores for custom TrAdaBoost implementation

Base Classifier	Model Score
Linear Discriminant Analysis	0.4861
Linear SVC	0.5972
MLP Classifier	0.7361
Gaussian Process Classifier	0.8125

Scores for Python Adapt TrAdaBoost

Iteration	SVM	SVMt	AUX	TrAdaBoost
Iteration 1	0.5694	0.5694	0.4583	0.5763
Iteration 2	0.5694	0.5625	0.4375	0.5833
Iteration 3	0.5694	0.5625	0.4791	0.5833
Iteration 4	0.5625	0.5694	0.5138	0.5763
Iteration 5	0.5694	0.5625	0.4861	0.5555
Iteration 6	0.5694	0.5694	0.5000	0.5555
Iteration 7	0.5694	0.5694	0.4652	0.5833
Iteration 8	0.5694	0.5694	0.5277	0.5555
Iteration 9	0.5694	0.5694	0.4791	0.5833
Iteration 10	0.5694	0.5694	0.4305	0.5416

Experimental errors for SVM, SVMt, AUX and Adapt TrAdaBoost over 10 iterations

	SVM	SVMt	AUX	TrAdaBoost
Error	0.562	0.567	0.478	0.569

Average of 10 repeats for SVM, SVMt, AUX and Adapt TrAdaBoost

6.2 Transfer Learning with Subspace Alignment:

Base Classifier	Python Adapt Score	Custom Implementation Score
MLP Classifier	0.2430	0.1527
K Neighbors	0.2013	0.2083
Linear SVC	0.2152	0.1388
SVC	0.1805	0.1875
Gaussian Process	0.2013	0.2152
Decision Tree	0.3819	0.2013
Random Forest	0.1944	0.1388
AdaBoost	0.1458	0.3541
Linear Discriminant Analysis	0.2430	0.0069

Scores for Subspace Alignment

6.3 Interpretation of TL Results:

6.3.1 TrAdaBoost:

- Table 1: Custom TrAdaBoost Implementation
  - The TrAdaBoost we implemented only predicts target labels 0 and 1 as the output regardless of the number of classes in the actual target labels.
  - This is a problem because our target labels range from 0 to 5.
    We could have solved this problem either by modifying our code further to account for multi-class classification, or modifying our dataset in a way that the classes in source, target and target test

are split into just two classes (grouping class A, B and C as class 0; grouping class D, E as class 1), performing TrAdaBoost and again unsplitting the data into their original classes.

- Given the time constraint of our project, we were unable to account for this. This is why in all three cases apart from baseline AdaBoost, the program halted in just one iteration causing the scores to be the same in all three cases.
- In all the cases, we believe the low scores are due to the extremely low number of datapoints for all datasets.
- An additional condition according to [11] was that the error rate must not exceed 0.5, which is also why our algorithm halted after 1-2 iterations.

- Table 2: Adapt TrAdaBoost Implementation
  - The TrAdaBoost model from the Adapt package on Python requires a base estimator, number of rounds (if gt or lt 10), target input and output data, learning rate (if != 1.0) etc. One way we can observe the different results from this model is by changing the base estimator or by modifying the learning rates. In our project, we have observed different scores with different base classifiers.
  - We shall analyse what the order of the scores (here, ascending) means.
    * The highest jump is between the Linear SVC and the MLP Classifier and we will address that first.
    * On observing the run time of both these classifiers for a number of iterations, it was observed that the MLPs run faster than the SVCs. This happens, from our inference, because SVC's training process involves solving the Lagrangian dual problem, rather than primal. This means that, for SVC, this is a quadratic optimization problem.
    * The increase in the performance of MLP is because these different classes cannot be separated by a single boundary (either linear or non-linear) but in fact need multiple boundaries to separate them. This is why MLPs will perform better than SVCs in our case.
    * Additional observation: If our model were to have a significantly larger dataset, the MLPs would perform even better due to this same reason as justified above. Which is, that the SVC will have to spend longer time searching for an optimum boundary if it is bound by the condition that there must be a single linear/non-linear boundary. Whereas MLPs will perform better in that case since MLPs would be easier to parallelize.
    * Linear SVC performs better than LDA since, in our opinion, LDA assumes that the data has the same covariance and the distribution of the points is assumed to be gaussian. Our data has has no such condition and this is why LDA performs poorly.
    * SVMs solve an optimization problem, whereas LDA solve an analytical problem on top of which the assumptions included with LDA make is less flexible.
    * Unfortunately, we do not have sufficient knowledge of the subject to accurately determine why the Gaussian Process Classifier performs best amongst the three other classifiers.

- Table 3 and 4: SVM, SVMt, AUX and Adapt TrAdaBoost
  - SVM refers to a linear SVM classifier fitted only with source data, SVMt with source and target labeled data, AUX refers to the Balanced Weighting method included in the code and TrAdaBoost using with a linear SVM classifier as the base learner.
  - We add a comparison to AdaBoost to verify that TrAdaBoost is not advantaged by the averaging of prediction over multiple estimators.
  - Table 3 indicates the different scores for all four over ten iterations and table 4 indicates the average error for all of them.
  - The experiment was performed over ten iterations with different random seeds and the trade-off parameter for AUX was set to 4 as per the authors of the Boosting for Transfer Learning [2] paper.

- The results that we obtained in this table were different from the ones obtained by the authors in the paper. This can be accounted for the significantly lesser number of datapoints that we have for source, target and target test datasets. We have 300 datapoints for all of them and two classes amongst these are quite unbalanced with respect to number.

6.3.2 Subspace Alignment:

- Table 5: Subspace Alignment Scores
  - In almost all of the scores, the Python Adapt score is just slightly higher or equal to the custom implementation score.
  - This could mainly be attributed to the lack of sufficient data, which means both the implementations require significantly larger datasets to perform well.

7 Contribution of each team member

Amulya Kallakuri: Transfer Learning
Nithyashree Manohar: Semi-supervised Learning

8 Summary and Conclusion

Next steps and findings for Semi-Supervised Learning:

It was found that most of the Semi-Supervised Learning algorithms perform better than the Supervised Learning algorithms. The next steps would be to implement QN-S3VM to a multi-class problem, implement other algorithms taught in class such as Harmonic Function algorithm, and Expectation Maximization algorithm.

The next steps for Transfer Learning would be as follows:

- Most importantly, increase the size of the dataset by a large amount so that predictions can be made better.
- The algorithm for TrAdaBoost will need to be modified to include multi-class classification in addition to binary. At the moment, we do not have the resources to account for multi-class classification, as per us and the TA Zhiruo Zhou.
- Our custom TrAdaBoost implementation ended in just 1-2 iterations and a good next step would be to figure out why this is happening, needless to state the obvious that is the lack of sufficient data samples.
- Deeper understanding of Gaussian Process Classifier would be essential to understand why and how it can give us better results.
- The custom implementation for Subspace Alignment can be improved by adding in more methods that perform some sort of data similarity or further dimensionality adjustment.
- Fine-tuning for transfer learning done in this paper was not particularly cut out for us since we did not have a pretrained model to work with. In the case that we have image or text data, fine tuning would be all the more useful. It is a suggestion of this paper that we may somehow implement the aspects of fine-tuning for these methods as well.

References

[1] Solar flare dataset for transfer learning from keel.

[2] Michelin Adapt and ENS Paris-Saclay Centre Borelli. Adapt feature based subspace alignment.

[3] Michelin Adapt and ENS Paris-Saclay Centre Borelli. Adapt instance based transfer learning.

[4] Dezhou Chen. Label propagation algorithm and its application, Jul 2021.

[5] Thiago Henrique Cupertino and Liang Zhao. Semi-supervised learning using random walk limiting probabilities. In *Advances in Neural Networks – ISNN 2013*, pages 395–404. Springer Berlin Heidelberg, 2013.

[6] Wenyuan Dai, Qiang Yang, Gui-Rong Xue, and Yong Yu. Boosting for transfer learning. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, page 193–200, New York, NY, USA, 2007. Association for Computing Machinery.

[7] Fabian Gieseke, Antti Airola, Tapio Pahikkala, and Oliver Kramer. Sparse quasi-newton optimization for semi-supervised support vector machines. In *International Conference on Pattern Recognition Applications and Methods*, 2012.

[8] Suraj Iyer. Scikit-learn style implementation of tradaboost algorithm.

[9] Alexander Ligthart, Cagatay Catal, and Bedir Tekinerdogan. Analyzing the effectiveness of semi-supervised learning approaches for opinion spam classification. *Applied Soft Computing*, 101:107023, 2021.

[10] S. Ozdemir and D. Susarla. *Feature Engineering Made Easy: Identify unique features from your dataset in order to build powerful machine learning systems.* Packt Publishing, 2018.

[11] Yi Yao and Gianfranco Doretto. Boosting for transfer learning with multiple sources. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1855–1862, 2010.

[12] Zhuanlan Zhihu. Transfer learning jda and tradaboost.