Analysis of Supervised and Semi-Supervised Machine Learning for Cervical Cancer Diagnosis

EE 660 Course Project

Maitreyee Likhite, mlikhite@usc.edu
Divya Nandlal Sahetya, sahetya@usc.edu

December 9, 2022

Project Type: (1) Design a system based on real-world data

1 Abstract

Cervical Cancer ranks as the fourth leading cause of death by cancer in women worldwide It has a very high possibility of cure when diagnosed in its early stages, but has only a few physiological symptoms. Therefore, risk factors such as age, number of pregnancies, and hormonal contraceptives, play a major role in the diagnosis using four tests for Cervical Cancer, namely Hinselmann, Schiller, Cytology and Biopsy. The objective of this project is to predict which diagnosis test results in the highest accuracy of Cervical Cancer Detection based on the 'Cervical Cancer (Risk Factors) Data Set.' Supervised Learning (SL) and Semi-Supervised Learning (SSL) are the two Machine Learning approaches used in this project for training different classifier models in Python's scikit-learn (sklearn) library, namely K-Nearest Neighbors (K-NN), Logistic Regression, Support Vector Machine (SVM), Decision Tree, Random Forest, and AdaBoost. These models are compared with the baseline trivial and non-trivial systems. The best classifier models for both, Supervised Learning and Semi-Supervised Learning, are determined based on the performance metrics in the sklearn.metrics module, namely accuracy score, f1-score, and precision. These best models for Supervised Learning and Semi-Supervised Learning approaches are also compared based on the accuracy score, to determine the best approach for Cervical Cancer diagnosis based on its risk factors.

2 Introduction

2.1 Problem Type, Statement and Goals

Type of the Problem: Multi-class Classification

Target variables (classes): Number of classes is 4 namely Hinselmann, Schiller, Cytology and Biopsy.

1. Class 1 - Hinselmann
   (a) Positive (1): 35 (4.08 %)
   (b) Negative (0): 823 (95.92 %)
2. Class 2 - Schiller
   (a) Positive (1): 74 (8.62 %)
   (b) Negative (0): 784 (91.37 %)
3. Class 3 - Cytology

   (a) Positive (1): 44 (5.13 %)
   (b) Negative (0): 814 (94.87 %)
4. Class 4 - Biopsy
   (a) Positive (1): 55 (6.40
   (b) Negative (0): 803 (93.60

Statement: The objective of this project is to predict which diagnosis test results in the highest accuracy of Cervical Cancer Detection based on the Cervical Cancer (Risk Factors) Dataset.

Problem: Cervical Cancer is a malignant gynecological tumor of the cervix, which is the lowermost part of the uterus, and ranks as the fourth leading cause of death by cancer in women worldwide, especially in the developing countries.

It still has a very high possibility of cure through chemotherapy, radiation or surgical treatment, when diagnosed in its early stages, but has only a few physiological symptoms such as pain or irregular bleeding.

Therefore, risk factors such as age, number of pregnancies and hormonal contraceptives, to name a few, play a major role in predicting the diagnosis, supported by the 4 tests for cervical cancer, namely Hinselmann, Schiller, Cytology and Biopsy.

Goals: The goals of this project is listed as follows:

- Process and clean the raw data to evaluate the contribution of all the risk factors (features) leading to the onset of Cervical Cancer.
- Detect if a person has the risk of cancer using the multi-class classification of the target labels (Hinselmann, Schiller, Cytology, Biopsy) and also which out of the four diagnosis tests is the best to detect cancer using baseline, SL and SSL models.
- Perform supervised Learning (SL) techniques like Logistic Regression, SVM, Random Forest Classifier and AdaBoost.
- Convert the SL dataset into a Semi-Supervised Learning (SSL) dataset by removing the target labels for, first 20% and then 60%, of the original labeled data.
- Compute the performance metrics (f1-score, accuracy and precision) to determine which test provides the most accurate detection. Compare the training accuracy and prediction performance of SSL with that of SL for the same classifier models used for SL [SSL dataset creation and performance evaluation].

Extension: The extension to this project is SSL. Details are as follows:

1. The dataset selected is a Supervised Learning (SL) dataset. One of the objective is to generate Semi-Supervised Learning (SSL) dataset from the SL dataset, by randomly (stratified sampling) eliminating the target values/labels, first for 20 % and then for 60 %, of the original data.
2. SSL techniques would then be implemented to classify whether a patient is having Cervical Cancer or not, and also to compare the SSL performance for different percentages of unlabeled data.
3. Finally, the performance of SSL will be compared with that of SL to determine the best ML approach for the particular problem and dataset.

Sources of difficulty (non-triviality) include:

1. The dataset is hard as it has a lot of features and requires a rigorous preprocessing.
2. High dimensionality of feature space.
   (a) Number of data points: 858
   (b) Number of features or input variables: 32
3. The data has has Missing Data as listed in the Table 1 below :

Table 1: Missing Data for each feature

	0
Age	0
Number of sexual partners	26
First sexual intercourse	7
Num of pregnancies	56
Smokes	13
Smokes (years)	13
Smokes (packs/year)	13
Hormonal Contraceptives	108
Hormonal Contraceptives (years)	108
IUD	117
IUD (years)	117
STDs	105
STDs (number)	105
STDs:condylomatosis	105
STDs:cervical condylomatosis	105
STDs:vaginal condylomatosis	105
STDs:vulvo-perineal condylomatosis	105
STDs:syphilis	105
STDs:pelvic inflammatory disease	105
STDs:genital herpes	105
STDs:molluscum contagiosum	105
STDs:AIDS	105
STDs:HIV	105
STDs:Hepatitis B	105
STDs:HPV	105
STDs: Number of diagnosis	0
STDs: Time since first diagnosis	787
STDs: Time since last diagnosis	787
Dx:Cancer	0
Dx:CIN	0
Dx:HPV	0
Dx	0
Hinselmann	0
Schiller	0
Citology	0
Biopsy	0

4. Limited number of training samples. The data is split in train and test in the ration 8:2 which further reduces the number of training samples.
5. Significant amounts of preprocessing required to handle missing data, data imbalance, outliers and not significant risk factors.

2.2 Literature Review

In paper [1], titled 'A Fully-Automated Deep Learning Pipeline for Cervical Cancer Classification', cervigrams, which are images of the cervical region, have been used to develop a fully-automated pipeline for cervix detection and evaluation of cervical cancer presence. Their proposed pipeline consists of two pre-trained deep learning models for the automatic cervix detection and cervical tumor classification. The first model detected the cervix region 1000 times faster than state-of-the-art data-driven models, while achieving a detection accuracy of 0.68 in terms of intersection of union (IoU) measure. Self-extracted features were used by the second model to classify the cervix tumors.

These features were learned using two lightweight models based on convolutional neural networks (CNN) [1].

In [2], titled 'Classification of Cervical Cancer Dataset', a study of feature selection, dimensionality reduction, and data balancing techniques that improve the classification accuracy for cervical cancer has been presented. Their key results include the main predictive features, a high accuracy of 97.5 %, and best performance shown by the Decision Tree Classifier.

2.3 Our Prior and Related Work - None

2.4 Overview of Our Approach

2.4.1 Main Topic: Supervised Learning (SL)

The baseline systems are:

1. The trivial baseline: A probability based classifier system that outputs class assignments S1 or S2 at random with probability N_1/N and N_2/N respectively has been defined as the trivial baseline system. N_i' is the population of data points with class label Si, and N is the total population of data points, based on the training set. The probability summarizes the likelihood of an data belonging to each class label.

   - This has been implemented using **Bernoulli distribution**:
     Class=0 (no cancer): P(class=1) = 0
     Class=1 (cancer): P(class=1) = 1
   - Bernoulli distribution was imported from scipy library [4].
   - The classifier was coded using object-oriented approach that has api similar to sklearn's estimator (it has fit, predict and score methods).

   The performance evaluation metric is accuracy score and the maximum test accuracy for each of the four classes (the four diagnosis tests for Cervical Cancer) can be highlighted as follows:

   - Hinselmann: 46.85 %
   - Schiller: 55.42 %
   - Cytology: 45.63 %
   - Biopsy: 44.62 %

   From the above results for the probability based random classifier, we infer that overall the random probability based classifier does not result in a good test accuracy, and that Schiller diagnosis test for Cervical Cancer has the highest test accuracy for the considered trivial baseline classifier.

2. The non-trivial baseline: 1-Nearest Neighbor (1-NN) classifier model using the KNeighborsClassifier module from the sklearn.neighbor library has been implemented as the non-trivial baseline system. The performance evaluation metric is accuracy score and the maximum test accuracy for each of the four classes (the four diagnosis tests for Cervical Cancer) can be highlighted as follows:

   - Hinselmann: 93.33 %
   - Schiller: 87.92 %
   - Cytology: 87.70 %
   - Biopsy: 92.43 %

   As additional analysis, the graph of Test Accuracy vs K was plotted for each class to evaluate the number of neighbors (K) resulting in the highest train and test accuracy. These are as follows:

- Hinselmann: Train Accuracy is maximum (99 %) for K = 1; Test Accuracy is maximum (93 %) for K = 2
- Schiller: Train Accuracy is maximum (99 %) for K = 1; Test Accuracy is maximum (88 %) for K = 2
- Cytology: Train Accuracy is maximum (99 %) for K = 1; Test Accuracy is maximum (88 %) for K = 1
- Biopsy: Train Accuracy is maximum (100 %) for K = 1; Test Accuracy is maximum (92 %) for K = 1

From the above results for 1-NN and K-NN (as additional analysis), we infer that Biopsy and Hinselmann are very accurate diagnosis tests for Cervical Cancer.

2.4.2 Extension: Semi-Supervised Learning (SSL)

The baseline system is: K-Nearest Neighbors

The K-Nearest Neighbor (K-NN) classifier model using the KNeighborsClassifier module from the sklearn.neighbor library has been implemented as the baseline system. The performance evaluation metric is accuracy score and the maximum test accuracy for each of the four classes (the four diagnosis tests for Cervical Cancer) can be highlighted as follows:

- Hinselmann: 93.33 %
- Schiller: 87.92 %
- Cytology: 87.70 %
- Biopsy: 92.43 %

From the above results for K-NN, we infer that Hinselmann is a very accurate diagnosis test for Cervical Cancer.
The baseline systems in both, SL and SSL, were compared with the classifier models of Logistic Regression, SVM, Decision Tree, Random Forest, XGBoost, and AdaBoost. The key performance metric used is the accuracy score. The best model was selected on the basis of comparison of the test accuracy with the accuracy of the baseline system.

The test data was separated before the data balancing step in preprocessing. Prior to training, hyperparameter tuning was performed by randomised grid-search cross validation using the RandomizedSearchCV() module in scikit-learn. The tuned parameters were used for training the models. The trained models were saved and loaded in the testing phase and the model resulting in the highest accuracy on the test data was selected as the best model. This process was followed for both approaches, SL and SSL.

For SSL, the original preprocessed dataset (used for SL) was converted to a dataset suitable for SSL by removing labels for certain number of data points. Both the labelled and unlabelled data points were used for training in SSL. Different percentages of unlabelled data were used, that is, 20 %, 40 %, and 60 %, so that the effect of the quantity of unlabelled data on the test accuracy of SSL can be analysed.

3 Implementation

3.1 Data Set

The dataset used in this project is the 'Cervical Cancer (Risk Factors) Data Set'. It has been downloaded from the UCI Machine Learning Dataset Repository and can be found at [3].

This dataset focuses on the prediction of indicators/diagnosis of Cervical Cancer. The features cover demographic information, habits, and historic medical records of 858 patients. The dataset was collected at 'Hospital Universitario de Caracas' in Caracas, Venezuela and was donated on March 3, 2017. Several patients decided not to answer some of the questions because of privacy concerns, leading to some missing values [3].

It is a multivariate dataset with integer and real attribute values. It has 858 instances (data points corresponding to each of the 858 patients), 32 attributes (input variables/features which are the risk factors for Cervical Cancer), and 4 target variables (classes corresponding to the 4 diagnosis tests for Cervical Cancer detection).

The problem domain for this project is Multi-class Classification and the chosen dataset is suitable for this particular problem as it allows us to predict the most accurate diagnosis test for Cervical Cancer, out of the four different tests, namely Hinselmann, Schiller, Citology, and Biopsy, based on 32 different risk factors. The features in the dataset and their corresponding data types have been listed in Table 2.

Table 2: Features and their Data Types

Feature	Data Type
Age:	int
Number of sexual partners:	int
First sexual intercourse (age):	int
Num of pregnancies:	int
Smokes:	bool
Smokes (years):	bool
Smokes (packs/year):	bool
Hormonal Contraceptives:	bool
Hormonal Contraceptives (years):	int
IUD:	bool
IUD (years):	int
STDs:	bool
STDs (number):	int
STDs:condylomatosis:	bool
STDs:cervical condylomatosis:	bool
STDs:vaginal condylomatosis:	bool
STDs:vulvo-perineal condylomatosis:	bool
STDs:syphilis:	bool
STDs:pelvic inflammatory disease:	bool
STDs:genital herpes:	bool
STDs:molluscum contagiosum:	bool
STDs:AIDS:	bool
STDs:HIV:	bool
STDs:Hepatitis B:	bool
STDs:HPV:	bool
STDs: Number of diagnosis:	int
STDs: Time since first diagnosis:	int
STDs: Time since last diagnosis:	int
Dx:Cancer:	bool
Dx:CIN:	bool
Dx:HPV:	bool
Dx:	bool
Hinselmann: target variable:	bool
Schiller: target variable:	bool
Cytology: target variable:	bool
Biopsy: target variable:	bool

3.2 Dataset Methodology

The steps involved in the usage of the dataset are listed as follows:

- Step 1: Train Test Split:

  The dataset is first divided into train and test data in the ratio 8:2. This is done to avoid data snooping and the split is done using stratified sampling. The number of data points in each class is listed in Table 3.

Table 3: Data count for train data

Hinselmann	Data points count
1	509
0	506
Total	1015
Schiller	Data points count
0	480
1	477
Total	957
Citology	Data points count
0	499
1	494
Total	993
Biopsy	Data points count
0	494
1	487
Total	981

Table 4: Data count for test data

Hinselmann	Data points count
1	127
0	127
Total	254
Schiller	Data points count
0	121
1	119
Total	240
Citology	Data points count
0	125
1	124
Total	249
Biopsy	Data points count
0	124
1	122
Total	246

- Step 2: Preprocessing:

  Preprocessing steps ranging from data cleaning to normalization is then performed on the train dataset. The detailed steps are explained in Section 3.3.

- Step 3: Conversion of SL dataset to SSL:

  Once the train data is cleaned, in order to be able to perform semi supervised learning on the dataset, some percentage of the labelled points is converted to unlabelled in a stratified fashion. So, the data will now have three classes 0 (labelled point indicating no cancer), 1 (labelled points indicating the presence of cancer) and -1 (indicating unlabelled points). The details of how many data points are in each class are listed in Table 5 and Table 6

Table 5: 20% unlabelled data

Hinselmann	Data points count
1	407
0	405
-1	203
Schiller	Data points count
0	384
1	381
-1	192
Citology	Data points count
0	399
1	395
-1	199
Biopsy	Data points count
0	395
1	389
-1	197

Table 6: 60% unlabelled data

Hinselmann	Data points count
-1	609
1	204
0	202
Schiller	Data points count
-1	596
0	199
1	198
Citology	Data points count
-1	596
0	199
1	198
Biopsy	Data points count
-1	589
0	197
1	195

- Step 4: Model Selection and Hyperparameter Tuning using Cross Validation:

  The labelled data points are used further for the model selection step using stratified k fold using grid search cross validation to select a model with the best hyperparameters. Here the number of folds(k) is set to 3.

- Step 5: Training for Supervised Models:

  The listed classifiers are trained with the best parameters using the labelled data points (train data).

- Step 6: Testing:

  The test data is used to predict whether the patient has cancer using the trained model for each of the classifiers.

3.3 Preprocessing and Exploratory Data Analysis (EDA)

- Preprocessing

1. Treating Missing Data: The columns (features/risk factors) and rows (feature values recorded for different patients) containing a large number **(more than 50 % of the entire data)** of missing values were identified. Those columns and rows were eliminated, while other missing values in a column were filled with the average of other values in that column (for int data) and with the previous value (for bool data). We observed that the features 'STDs: Time since first diagnosis' and 'STDs: Time since last diagnosis' both had **787** missing values, which is the highest number of missing values among all the features. Therefore, these 2 features were removed from the dataset. Similarly, we observed that there are 2 patients whose individual data records had 22 missing values and 3 patients whose individual data records had 23 missing values. Therefore, the data of these patients could be removed from the data set due to the high number of missing feature values.
2. Data Type Casting: The binary categorical data was type cast by converting all int and bool data values to the float data type, for convenience of further computation in ML models.
3. Removing Outliers: By plotting the box plot of all the features (detailed in Step 2 under EDA), it was identified that the features **'Age'** and **'Number of sexual partners'** had the maximum number of outliers. Hence, for the feature 'Age', data values **greater than 52** (threshold for age feature) were eliminated and for the feature 'Number of sexual partners', data values **greater than 8** (threshold for the particular feature) were eliminated.
4. Computing Class Imbalance: For each of the 4 classes, namely Hinselmann, Schiller, Cytology, and Biopsy, the number of data samples having target value 0 and target value 1 were counted. A particular target variable is said to be imbalanced when it contains a higher number of data samples belonging to one class as compared to another class. Computing the imbalance for all 4 classes enabled to identify this lack of uniformity and was rectified by a data balancing algorithm (step 9 of preprocessing) after separating the test data.
5. Creating Data Ranges: The 'Age' feature was converted into **'age_group'** with Group 1 having ages from 10 to 14, Group 2 having ages from 15 to 19, and so on till Group 9 having ages 50 to 54. The minimum age was set to 13 and the maximum age was set to 52 after removing outliers. Similarly, the 'First sexual intercourse age' feature was converted into **'fsi_group'** with Group 1 having ages from 10 to 14, Group 2 having ages from 15 to 19, and so on till Group 5 having ages 30 to 34. The minimum age for this feature was set to 10 and the maximum age was set to 32 after removing outliers.
6. Removing Features having a Single Unique Value: The features **'STDs:cervical condylomatosis'** and **'STDs:AIDS'** were found to have only a single unique value in their respective columns, hence these features were eliminated from the dataset as they would not contribute significantly to any single target class.
7. Data Regularization/Normalization: The numeric data values were normalized to a feature range of **(1, 2)** using the **MinMaxScaler()** module of the sklearn.preprocessing library. This regularization transform was applied to the feature vector using the **fit_transform()** module of the sklearn.preprocessing.StandardScaler library. The histograms of all the data features after normalization are shown in Figure 1.

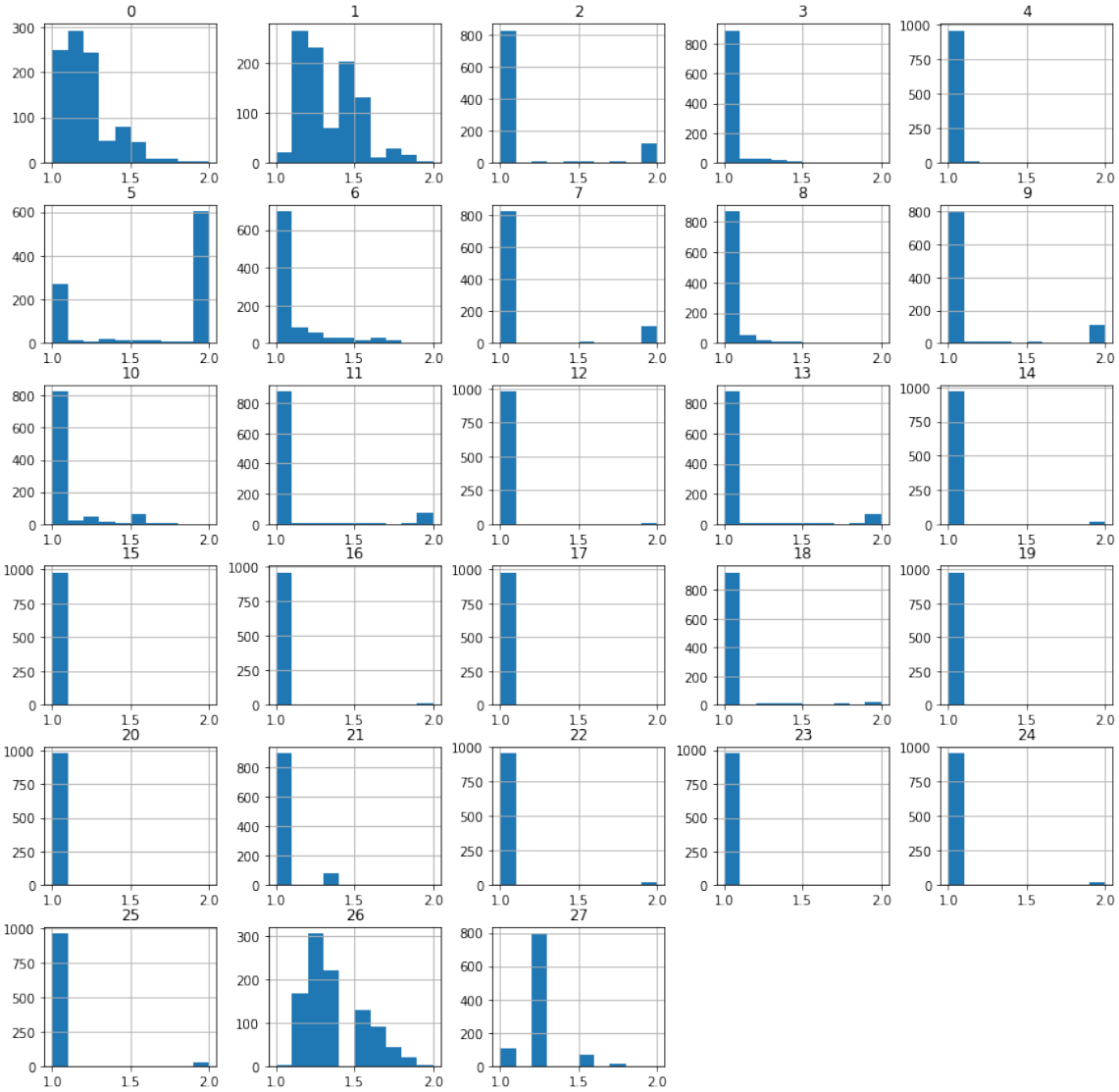

Figure 1: Histograms of all the Data Features after Normalization

8. Dataset Bifurcation: The original dataset was split into a training dataset (**60 %**, for preliminary data analysis and training the baseline models and ML models), a validation dataset (**20 %**, k-fold cross-validation on all the ML models was implemented), and a randomly sampled, unused and unseen test dataset (**20 %** using stratified sampling, for final prediction and performance evaluation on the best model).
9. Data Balancing: Data augmentation was implemented for the minority class using the Synthetic Minority Oversampling Technique **(SMOTE)** that duplicates data points in the minority class or synthesizes new examples from the existing data points.

- EDA

1. Feature Correlation Heatmap: Depending on the correlation matrix between the input features and the output target classes, the features having a correlation of more than 50% with the output would only be selected and considered for further processing. The correlation heatmap is shown in Figure 2.

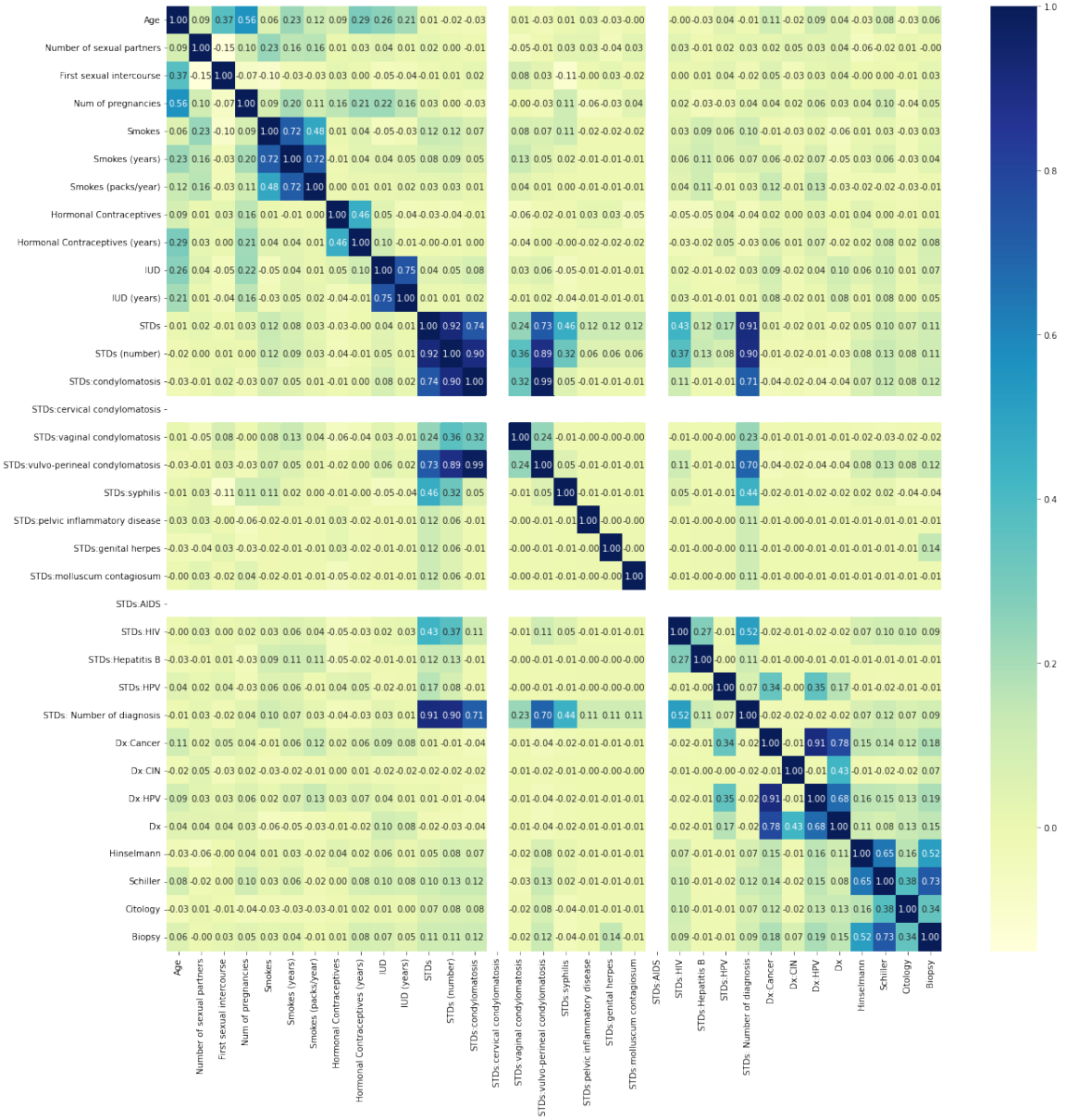

Figure 2: Correlation Heatmap

2. Box Plot : The box plot was plotted using the box_plot() module of the matplotlib.pyplot library. The distribution of features was observed using the box plot and this information was utilized when balancing the target classes. The box plot of the original data features is shown in Figure 3.

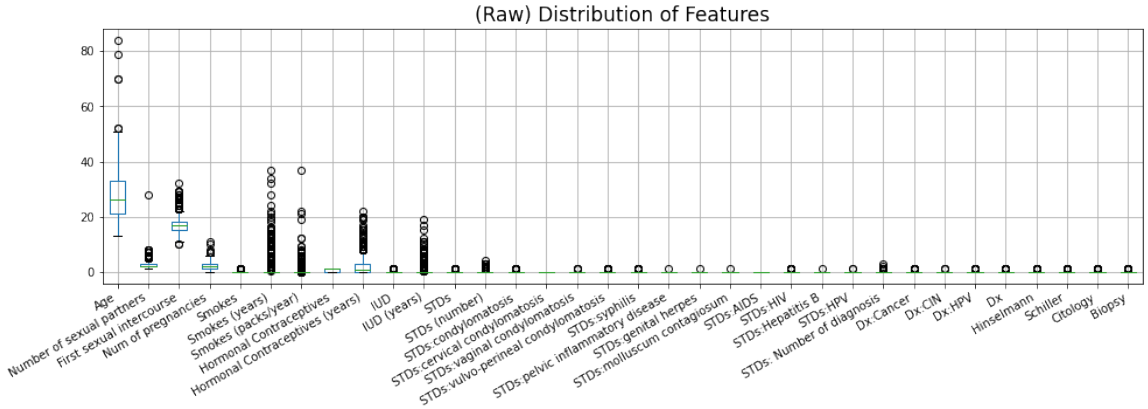

Figure 3: Distribution of Original Data Features in a Box Plot

3. Statistical Analysis: The mode and median for all the features was computed using the mode() and median() modules of the statistics library. The mode is that data point for any feature at which the probability density function of the feature has a local maximum. The median separates the higher half of the data values for a particular feature from the lower half.

3.4 Training Process

- The training data is then passed to the training stage where each of the classifier is trained with the best parameters and the trained model is stored in pickled.

- **Sample Complexity:** The sample complexity denotes how many training samples N are needed to achieve a certain generalization performance. The performance is specified by two parameters, ϵ and δ. The error tolerance ϵ determines the allowed generalization error, and the confidence parameter δ determines how often the error tolerance is violated. How fast N grows as ϵ and δ become smaller indicates how much data is needed to get good generalization.

  The VC bound can be used to estimate the sample complexity for a given learning model. By fixing $\delta > 0$, to get generalization error to be at most ϵ. From Equation 1, the generalization error is bounded by $\sqrt{\frac{8}{N} \ln \frac{4m_h(2N)}{\delta}}$ and so it makes $\sqrt{\frac{8}{N} \ln \frac{4m_h(2N)}{\delta}} \leq \epsilon$. It follows that, $N \geq \frac{8}{N} \ln \frac{4m_h(2N)}{\delta}$ suffices to obtain generalization error at most ϵ (with probability at least 1- δ).

$$E_{out}(g) \leq E_{in}(g) + \sqrt{\frac{8}{N} \ln \frac{4m_h(2N)}{\delta}} \tag{1}$$

  For example, in case of **Hinselman Test for 20% unlabelled data**,

  D (D features) = 28

  N (training points) = 812 (407+405)

  Expressing the upper bound on the out-of-sample error as $E_{out}(h_g) \leq E_{in}(h_g) + \epsilon_{vc}$. For $E_{in}(h_g)$ measures on training data, d_{vc} is used to get the value ϵ_{vc}.

  Therfore,

$$E_{eff} \leq \sqrt{\frac{8}{N} \ln \frac{4m_h(2N)}{\delta}} = E_{vc} = \sqrt{\frac{8}{N} \ln \frac{4((2N)^{d_{vc}} + 1)}{\delta}} \tag{2}$$

  Therefore, $\epsilon_{vc} = 1.465$.To get a lower ϵ_{vc} the number of features, D should be reduced and N should be increased to get a bound closer to 0.1 for training data.

3.4.1 Baseline System - Trivial

- A system that outputs class assignments S1 or S2 at random with probability N_1/N and N_2/N, respectively; N_i' is the population of data points with class label Si, and N is the total population of data points, based on the training set. The probability summarizes the likelihood of an data belonging to each class label, hence has been chosen as the trivial baseline system. When an data reading has the class label **no cancer** (0), then the probability of the class labels 0 and 1 will be 1 and 0 respectively and when an data reading has the class label **cancer** (1), then the probability of class labels 0 and 1 will be 0 and 1 respectively.
- Class=0 (no cancer): P(class=0) = 1, P(class=1) = 0

  Class=1 (cancer): P(class=0) = 0, P(class=1) = 1
- This has been implemented using **Bernoulli distribution**:

  Class=0 (no cancer): P(class=1) = 0

  Class=1 (cancer): P(class=1) = 1
- Bernoulli distribution was imported from scipy library [4].
- The classifier was coded using object-oriented approach that has api similar to sklearn's estimator (it has fit, predict and score methods).

3.4.2 Baseline System - Non Trivial - KNN

K-NN algorithm assumes the similarity between the new data and available cases and puts the new case into the class that is most similar to the available classes. It is a simple classifier, hence has been chosen as a non-trivial baseline to compare the final best-performing models.

Algorithm of KNN:

- Step-1: Select the number K of the neighbors
- Step-2: Calculate the Euclidean distance of K number of neighbors
- Step-3: Take the K nearest neighbors as per the calculated Euclidean distance.
- Step-4: Among these k neighbors, count the number of the data points in each class.
- Step-5: Assign the new data points to that class for which the number of the neighbor is maximum.
- KNN Classifier was implemented using sklearn function KNeighborsClassifier [5].

Table 7: KNN hyper-parameters

	n_neighbours
0	2 to 30

Algorithm 1 Algorithm (pseudocode) for KNN training

Require: Trainval is available with below preprocessing:

- Standardization and/or Normalization,
- Feature reduction (correlation, outliers, missing values)

```
function FIT(k_neighbours, trainval_data, trainval_labels)
    best_params = 1
    params_vs_score = {}
    for k in k_neighbours do
        scores = []
        for train, val from CrossValidation do
            KNeighborsClassifier(k=k)
            model.fit(train)
            curr_score = model.score(val)
            scores.append(curr_score)
        end for
        params_vs_score[hp] = Average(scores)
    end for
    best_params = k where params_vs_score has max(average accuracy)
    model.set_params(**best_params)
    pickle(model)
end function
```

3.4.3 Logistic Regression Classifier

- Logistic Regression can be used as a classification algorithm when the value of the target variable is categorical in nature, that is when it has a binary output or belongs to one class or another, or is either a 0 or 1. Hence, this model could be suitable to the Cervical Cancer dataset with all 4 output classes (Hinselmann, Schiller, Cytology, Biopsy) being binary target variables.
- Sigmoid function is used to map the predicted values to probabilities as it maps real value into another value between 0 to 1.
- Logistic Regression Classifier was implemented using sklearn function LogisticRegression [6].
- The parameters is chosen by model selection (Stratified K fold using GridSearchCV [7]). The details of the parameters are listed in Table 8.

Algorithm 2 Algorithm (pseudocode) for Logistic Regression training

Require: Trainval is available with below preprocessing:

- Standardization and/or Normalization,
- Feature reduction (correlation, outliers, missing values)

```
    params ='penalty': ['l1','l2'], 'C':np.logspace(-3,3,7)
    model_list = list of LogisticRegression models of different combination of params
function FIT(model_list, hyperparameters, trainval_data, trainval_labels)
    best_params = {}
    for model in model_list do
        params_vs_score = {}
        for hp in hyperparameters do
            scores = []
            for train, val from CrossValidation do
                model.fit(train)
                curr_score = model.score(val)
                scores.append(curr_score)
            end for
            params_vs_score[hp] = Average(scores)
        end for
        best_params[mo del] = hp where params_vs_score has max(average accuracy)
    end for
end function
```

Table 8: Logistic Regression Classifier parameters

	penality
0	l1
1	l2
	C
0	0.001
1	0.010
2	0.100
3	1.000
4	10.000
5	100.000
6	1000.000

3.4.4 SVM Classifier

- The main objective of the SVM Classifier is to classify or segregate a dataset with the best possible accuracy. Therefore, it is hypothesized that it could perform well to solve the multi-class classification problem of the selected dataset. The distance between nearest points on either side of the classification/decision boundary is known as the margin.
- The goal is to generate a hyperplane with the maximum possible margin value between support vectors of a dataset.
- The SVM Classifier was implemented using the SVC() module of the sklearn.svm library. The implementation is based on libsvm. The fit time scales at least quadratically with the number of samples. The multiclass support is handled according to a one-vs-one scheme [8].

Algorithm 3 Algorithm (pseudocode) for SVC training

Require: Trainval is available with below preprocessing:

- Standardization and/or Normalization,
- Feature reduction (correlation, outliers, missing values)

```
    params = "kernel":["rbf", "poly"], "gamma": ["auto", "scale"], "degree":range(1,6,1)
    model_list = list of SVC models of different combination of params
function FIT(model_list, hyperparameters, trainval_data, trainval_labels)
    best_params = {}
    for model in model_list do
        params_vs_score = {}
        for hp in hyperparameters do
            scores = []
            for train, val from CrossValidation do
                model.fit(train)
                curr_score = model.score(val)
                scores.append(curr_score)
            end for
            params_vs_score[hp] = Average(scores)
        end for
        best_params[mo del] = hp where params_vs_score has max(average accuracy)
    end for
end function
```

Table 9: SVC parameters

	kernel
0	rbf
1	poly
	degree
0	1
1	2
2	3
3	4
4	5
	gamma
0	auto
1	scale

3.4.5 Decision Tree Classifier

- Decision Tree is a Supervised Machine Learning Algorithm that uses a set of rules to make decisions, similarly to how humans make decisions. The intuition behind Decision Trees is that we use the dataset features to create yes/no questions and continually split the dataset until we isolate all data points belonging to each class. With this process, we are organizing the data in a tree structure [9]. The Decision Tree model is known to perform well for high dimensional data, hence has been chosen for the particular dataset having 28 features.

- Every time we ask a question, we are adding a node to the tree, and the first node is called the root node. The result of asking a question splits the dataset based on the value of a feature, and creates new nodes. If we decide to stop the process after a split, the last nodes created are called leaf nodes [9].

- The algorithm tries to completely separate the dataset such that all leaf nodes, i.e., the nodes that don't split the data further, belong to a single class. These are called pure leaf nodes, but most times we end up with mixed leaf nodes, where not all data points belong to the same class [9].

- In the end, the algorithm can only assign one class to the data points in each leaf node. With pure leaf nodes that already taken care of, because all data points in that node have the same class. But with mixed leaf nodes the algorithm assigns the most common class among all data points in that node [9].

- The Decision Tree Classifier has been implemented using the DecisionTreeClassifier() module of the sklearn.tree library [10].

Algorithm 4 Algorithm (pseudocode) for Decision Tree training

Require: Trainval is available with below preprocessing:

- Standardization and/or Normalization,
- Feature reduction (correlation, outliers, missing values)

```
    params = 'max_leaf_nodes': list(range(2, 100)), 'min_samples_split': [2, 3, 4]
    model_list = list of DecisionTreeClassifier() models of different combination of params
function FIT(model_list, hyperparameters, trainval_data, trainval_labels)
    best_params = {}
    for model in model_list do
        params_vs_score = {}
        for hp in hyperparameters do
            scores = []
            for train, val from CrossValidation do
                model.fit(train)
                curr_score = model.score(val)
                scores.append(curr_score)
            end for
            params_vs_score[hp] = Average(scores)
        end for
        best_params[model] = hp where params_vs_score has max(average accuracy)
    end for
end function
```

Table 10: Decision Tree parameters

	max_leaf_nodes
0	2 to 100
	min_samples_split
1	2
2	3
3	4

3.4.6 Random Forest Classifier

- The Random forest classifier creates a set of decision trees from a randomly selected subset of the training set. It is basically a set of decision trees from a randomly selected subset of the training set and then It collects the votes from different decision trees to decide the final prediction [11]. The Random Forest model is known to perform well for multi-dimensional data, hence has been chosen for the particular dataset having 28 features.
- A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting [12].
- The Random Forest Classifier has been implemented using the RandomForestClassifier() module of the sklearn.ensemble library, wherein the sub-sample size is controlled with the max_samples parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree [12].

Algorithm 5 Algorithm (pseudocode) for Random Forest training

Require: Trainval is available with below preprocessing:

- Standardization and/or Normalization,
- Feature reduction (correlation, outliers, missing values)

params = 'n_estimators': [200, 500], 'max_features': ['auto', 'sqrt', 'log2'], 'max_depth' : [4,5,6,7,8], 'criterion' :['gini', 'entropy']
model_list = list of RandomForestClassifier() models of different combination of params

```
function FIT(model_list, hyperparameters, trainval_data, trainval_labels)
    best_params = {}
    for model in model_list do
        params_vs_score = {}
        for hp in hyperparameters do
            scores = []
            for train, val from CrossValidation do
                model.fit(train)
                curr_score = model.score(val)
                scores.append(curr_score)
            end for
            params_vs_score[hp] = Average(scores)
        end for
        best_params[model] = hp where params_vs_score has max(average accuracy)
    end for
end function
```

Table 11: Random Forest parameters

	n_estimators
0	200
1	500
	max_features
0	auto
1	sqrt
2	log2
	max_depth
0	2-4
	criterion
0	gini
1	entropy

3.4.7 AdaBoost Classifier

- AdaBoost is an iterative ensemble method that combines multiple classifiers to increase the accuracy of classifiers. The underlying principle of AdaBoost is that it sets the weights of classifiers and trains the data samples in each iteration such that it ensures the accurate predictions of unusual observations [13]. Therefore, this model could perform well for the Cervical Cancer dataset.

- Any machine learning algorithm can be used as a base classifier if it accepts weights on the training set. AdaBoost requires two conditions:

  – The classifier should be trained interactively on various weighted training examples.

  – In each iteration, it tries to provide an excellent fit for these examples by minimizing training error [13].

- The AdaBoost Classifier has been implemented using the AdaBoostClassifier of the sklearn.ensemble library. This class implements the algorithm known as AdaBoost-SAMME. It is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases [14].

Table 12: AdaBoost Classifier parameters

	base_estimator
0	Decision Tree
	n_estimators
0	10
1	50
. 2	250
3	1000
	learning_rate
0	0.01
0	0.1

Algorithm 6 Algorithm (pseudocode) for AdaBoostClassifier training

Require: Trainval is available with below preprocessing:

- Standardization and/or Normalization,
- Feature reduction (correlation, outliers, missing values)

params = 'base_estimator':[DecisionTreeClassifier(random$_state$ = 42)],$'n\_estimators'$: $[10, 50, 250, 1000]$,$'learning\_rate'$: $[0.01, 0.1]$

model_list = list of AdaBoostClassifier() models of different combination of params

function FIT(model_list, hyperparameters, trainval_data, trainval_labels)

```
best_params = {}
for model in model_list do
    params_vs_score = {}
    for hp in hyperparameters do
        scores = []
        for train, val from CrossValidation do
            model.fit(train)
            curr_score = model.score(val)
            scores.append(curr_score)
        end for
        params_vs_score[hp] = Average(scores)
    end for
    best_params[model] = hp where params_vs_score has max(average accuracy)
end for
```

end function

3.5 Model Selection and Comparison of Results

For model selection we begin with preprocessed (Feature engineered, Standardize and/or Normalized and Feature Selection or Reduction applied) data in the start. We have a HyperParameterTuning function that takes list of models hyperparameters and trainval data and returns a dictionary of best parameters for each model as its key and hyperparameter and avg accuracy on crossvalidation set. For CrossValidation strategy we have tried StratifiedKFold available in sklearn and also tried some similar implementation of our own. We used split size of 3 for the CrossValidation. Steps involved are:

- Step1 :cv = GridSearchCV(pipe, clf_params,cv=cv)
- Step2: cv.fit(features,labels)
- Step3: clf_params will be expanded to get all possible combinations separate using ParameterGrid. features will now be split into features_train and features_val using cv. Same for labels
- Step4: Now the gridSearch estimator (pipe) will be trained using features_train and labels_inner and scored using features_test and labels_val.
- Step5: For each possible combination of parameters in step 3, The steps 4 and 5 will be repeated for cv_iterations. The average of score across cv iterations will be calculated, which will be assigned to that parameter combination. This can be accessed using cv_results_ attribute of gridSearch.

Algorithm 7 Algorithm (pseudocode) for Model selection and hyperparameter tuning

Require: Trainval and Test set is available with below preprocessing:

- Standardization and/or Normalization,
- Feature reduction (correlation, outliers, missing values)

```
function HyperParameterTuning(model_list, hyperparameters, trainval_data, trainval_labels)
    best_params = {}
    for model in model_list do
        params_vs_score = {}
        for hp in hyperparameters do
            scores = []
            for train, val from CrossValidation do
                model.fit(train)
                curr_score = model.score(val)
                scores.append(curr_score)
            end for
            params_vs_score[hp] = Average(scores)
        end for
        best_params[mo del] = hp where params_vs_score has max(average accuracy)
    end for
end function
```

3.5.1 Hinselman Test

- Train data with 20% and 60% is used and best hyperparameters are selected for the respective dataset.
- Table 13 lists down the best hyperparameters for the Hinselman Test for 20% unlabelled data. Since this is a supervised learning only the labelled datapoints in considered after converting dataset from SL to SSL.
- Table 14 lists down the average training accuracy over the 3 cross validation folds.
- Figure 4 depicts the decision boundary for k=2 best features of the model trained with labelled datapoints of the dataset where 20% of the data is unlabelled.
- Similar steps was followed for dataset wherein 60% of the datapoints was masked to get unlabelled points.

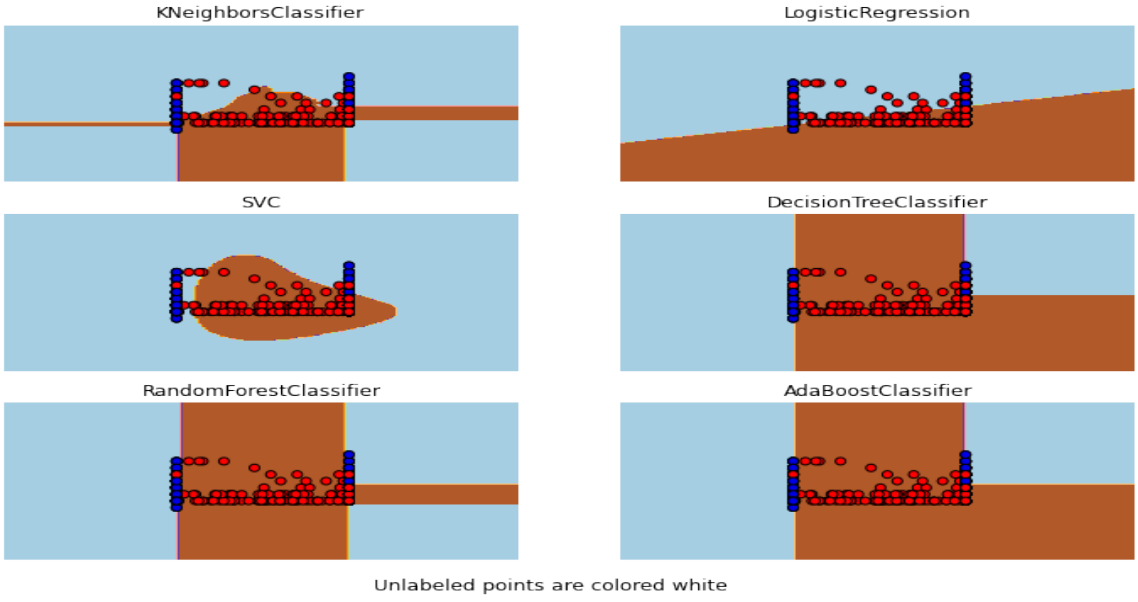

Figure 4: SL Hinselman Test Decision Boundary on top 2 best features using Kbest

Table 13: Best Hyper- parameters for Hinselman Test

KNeighborsClassifier	
n_estimators	2
LogisticRegression	
C	1000.0
penalty	l2
SVC	
kernel	poly
DecisionTreeClassifier	
max_leaf_nodes	85
min_samples_split	3
RandomForestClassifier	
criterion	gini
max_depth	8
max_features	sqrt
n_estimators	500
AdaBoostClassifier	
base_estimator	DecisionTreeClassifier
learning_rate	0.1
n_estimators	50

Table 14: Average accuracy over multiple cross-validation runs for Hinselmann Test

k folds	3	
KNeighborsClassifier	Accuracy(%)	E_{in}
	82.77%	0.1723
LogisticRegression	Accuracy(%)	E_{in}
	75.61%	0.2439
SVC	Accuracy(%)	E_{in}
	81.78%	0.1822
DecisionTreeClassifier	Accuracy(%)	E_{in}
	84.73%	0.1527
RandomForestClassifier	Accuracy(%)	E_{in}
	89.41%	0.1059
AdaBoostClassifier	Accuracy(%)	E_{in}
	88.66%	0.1134

3.5.2 Schiller Test

- Train data with 20% and 60% is used and best hyperparameters are selected for the respective dataset.
- Table 15 lists down the best hyperparameters for the Schiller Test for 20% unlabelled data. Since this is a supervised learning only the labelled datapoints in considered after converting dataset from SL to SSL.
- Table 16 lists down the average training accuracy over the 3 cross validation folds.
- Figure 5 depicts the decision boundary for k=2 best features of the model trained with labelled datapoints of the dataset where 20% of the data is unlabelled.
- Similar steps was followed for dataset wherein 60% of the datapoints was masked to get unlabelled points.

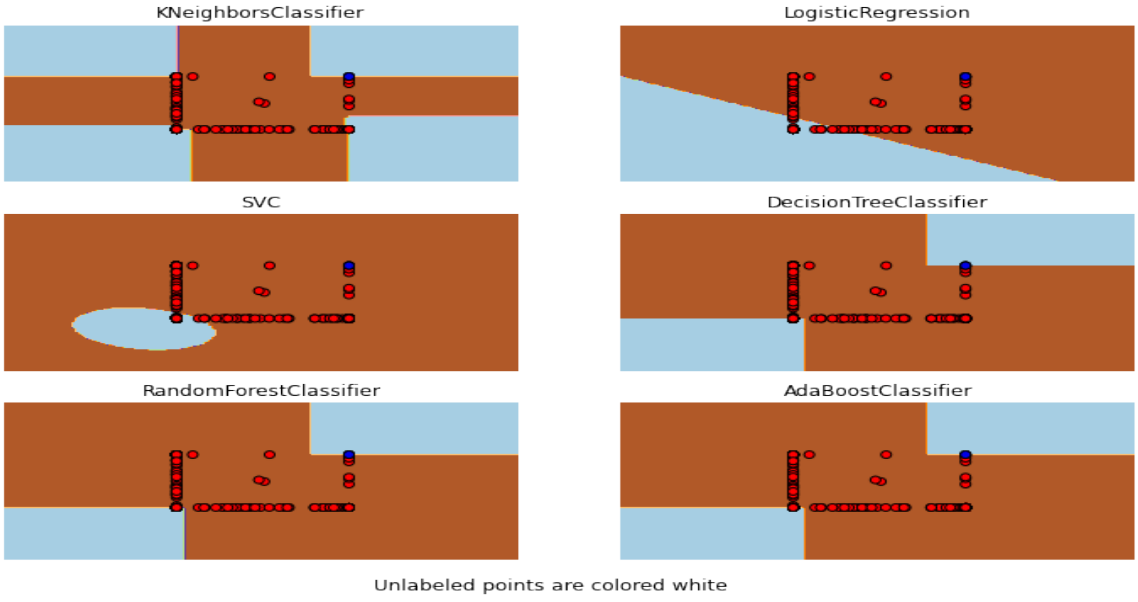

Figure 5: SL Schiller Test Decision Boundary on top 2 best features using Kbest

Table 15: Best Hyper- parameters for Schiller Test

KNeighborsClassifier	
n_estimators	2
LogisticRegression	
C	10.0
penalty	l2
SVC	
kernel	poly
DecisionTreeClassifier	
max_leaf_nodes	90
min_samples_split	4
RandomForestClassifier	
criterion	gini
max_depth	8
max_features	auto
n_estimators	500
AdaBoostClassifier	
base_estimator	DecisionTreeClassifier
learning_rate	0.1
n_estimators	1000

Table 16: Average accuracy over multiple cross-validation runs for Schiller Test

k folds	3	
KNeighborsClassifier	Accuracy(%)	E_{in}
	77.51%	0.2249
LogisticRegression	Accuracy(%)	E_{in}
	67.81%	0.3219
SVC	Accuracy(%)	E_{in}
	77.50%	0.2250
DecisionTreeClassifier	Accuracy(%)	E_{in}
	84.82%	0.1518
RandomForestClassifier	Accuracy(%)	E_{in}
	84.58%	0.1542
AdaBoostClassifier	Accuracy(%)	E_{in}
	83.51%	0.1649

3.5.3 Cytology Test

- Train data with 20% and 60% is used and best hyperparameters are selected for the respective dataset.
- Table 17 lists down the best hyperparameters for the Cytology Test for 20% unlabelled data. Since this is a supervised learning only the labelled datapoints in considered after converting dataset from SL to SSL.
- Table 18 lists down the average training accuracy over the 3 cross validation folds.
- Figure 6 depicts the decision boundary for k=2 best features of the model trained with labelled datapoints of the dataset where 20% of the data is unlabelled.
- Similar steps was followed for dataset wherein 60% of the datapoints was masked to get unlabelled points.

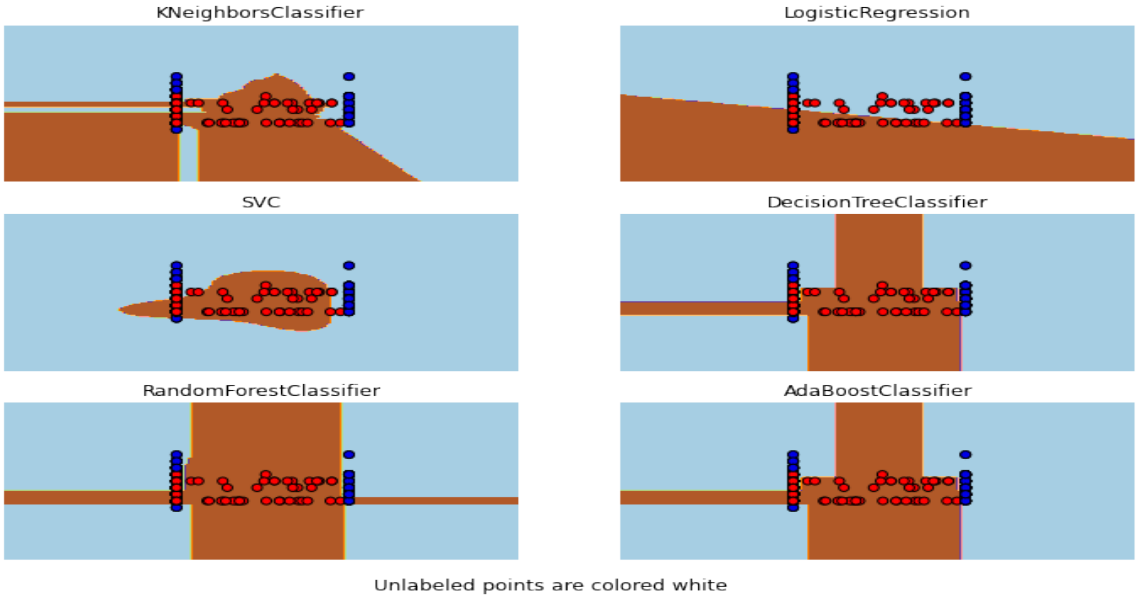

Figure 6: SL Cytology Test Decision Boundary on top 2 best features using Kbest

Table 17: Best Hyper- parameters for Cytology Test

KNeighborsClassifier	
n_estimators	2
LogisticRegression	
C	1000.0
penalty	l2
SVC	
kernel	poly
DecisionTreeClassifier	
max_leaf_nodes	69
min_samples_split	3
RandomForestClassifier	
criterion	gini
max_depth	8
max_features	auto
n_estimators	200
AdaBoostClassifier	
base_estimator	DecisionTreeClassifier
learning_rate	0.1
n_estimators	1000

Table 18: Average accuracy over multiple cross-validation runs for Cytology Test

k folds	3	
KNeighborsClassifier	Accuracy(%)	E_{in}
	76.59%	0.2341
LogisticRegression	Accuracy(%)	E_{in}
	68.53%	0.3147
SVC	Accuracy(%)	E_{in}
	75.33%	0.2467
DecisionTreeClassifier	Accuracy(%)	E_{in}
	81.10%	0.1890
RandomForestClassifier	Accuracy(%)	E_{in}
	86.39%	0.1361
AdaBoostClassifier	Accuracy(%)	E_{in}
	87.40%	0.1260

3.5.4 Biopsy Test

- Train data with 20% and 60% is used and best hyperparameters are selected for the respective dataset.
- Table 19 lists down the best hyperparameters for the Biopsy Test for 20% unlabelled data. Since this is a supervised learning only the labelled datapoints in considered after converting dataset from SL to SSL.
- Table 20 lists down the average training accuracy over the 3 cross validation folds.
- Figure 7 depicts the decision boundary for k=2 best features of the model trained with labelled datapoints of the dataset where 20% of the data is unlabelled.
- Similar steps was followed for dataset wherein 60% of the datapoints was masked to get unlabelled points.

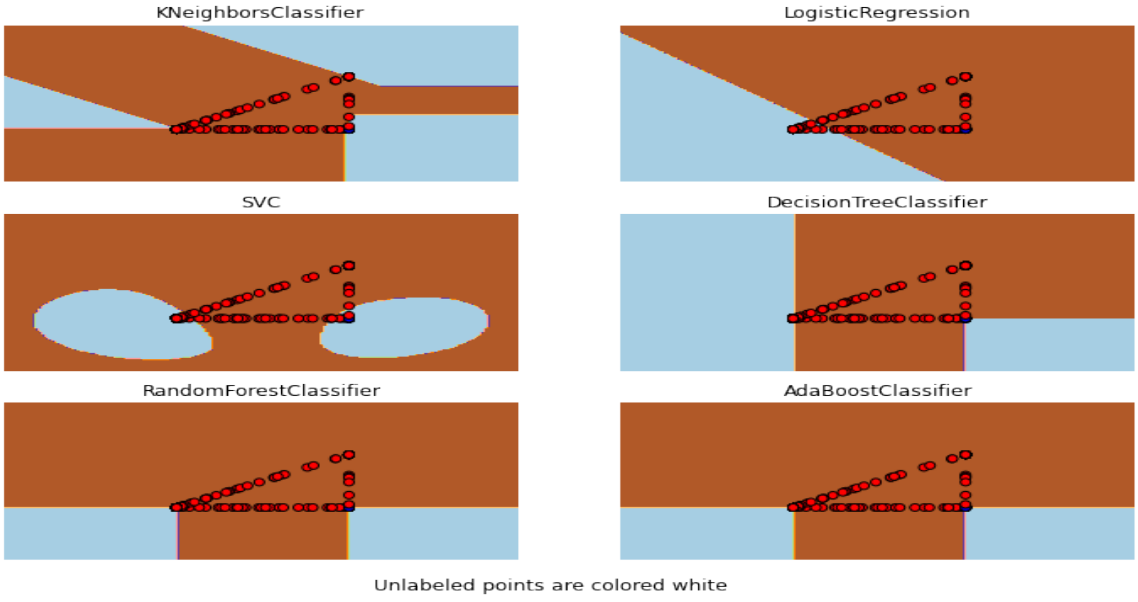

Figure 7: SL Biopsy Test Decision Boundary on top 2 best features using Kbest

Table 19: Best Hyper- parameters for Biopsy Test

KNeighborsClassifier	
n_estimators	2
LogisticRegression	
C	1000.0
penalty	l2
SVC	
kernel	poly
DecisionTreeClassifier	
max_leaf_nodes	78
min_samples_split	2
RandomForestClassifier	
criterion	gini
max_depth	8
max_features	auto
n_estimators	500
AdaBoostClassifier	
base_estimator	DecisionTreeClassifier
learning_rate	0.1
n_estimators	1000

Table 20: Average accuracy over multiple cross-validation runs for Biopsy Test

k folds	3	
KNeighborsClassifier	Accuracy(%)	E_{in}
	82.66%	0.1734
LogisticRegression	Accuracy(%)	E_{in}
	57.14%	0.4286
SVC	Accuracy(%)	E_{in}
	80.11%	0.1989
DecisionTreeClassifier	Accuracy(%)	E_{in}
	86.74%	0.1326
RandomForestClassifier	Accuracy(%)	E_{in}
	88.27%	0.1173
AdaBoostClassifier	Accuracy(%)	E_{in}
	85.47%	0.1453

4 Final Results and Interpretation

The testing was performed on the model trained with labelled data points after partitioning 20% of the data to unlabelled trained model and similarly for 60%.

The performance metric to choose the best model, tests (Hinselmann, Schiller, Cytology, Biopsy) or classifier is selected using the **accuracy** metric.

Steps involved are :

- Step 1: Select 20% trained models
- Step 2: For each of the classifiers (KNeighborsClassifier, LogisticRegression, SVC, DecisionTreeClassifier, RandomForestClassifier, AdaBoostClassifier), select the test which gives the highest accuracy. The highest accuracy helps us conclude that "this test" is the best for this "classifier" to predict if the patient has cancer or not based on the risk factors.
- Step 3: Once, the best test is selected, then the best classifier is selected.
- Step 4: Repeat Steps 1 to 3 for 60% trained models.
- Step 5: Compare the results of 20% and 60% models.

4.1 Testing Accuracy for 20% unlabelled Data

The Table 21 provides the accuracy score report for each of the classifier against each of the tests. On finding the target class having the maximum test accuracy for each of the classifiers in the code, we obtained the result as follows:

- For KNeighborsClassifier, the best test is: Hinselmann
- For LogisticRegression, the best test is: Hinselmann
- For SVC, the best test is: Hinselmann
- For DecisionTreeClassifier, the best test is: Hinselmann
- For RandomForestClassifier, the best test is: Hinselmann
- For AdaBoostClassifier, the best test is: Hinselmann

Therefore, Hinselmann test is the best test. Figure 8 shows the accuracy score for each of the classifier for Hinselmann test.

Table 21: Summary of Testing accuracy

	Hinselmann	Schiller	Citology	Biopsy
LogisticRegression	0.6535	0.6083	0.6345	0.6179
SVC	0.815	0.775	0.8072	0.7033
DecisionTreeClassifier	0.9016	0.675	0.6064	0.6382
RandomForestClassifier	0.8504	0.7875	0.7711	0.8211
AdaBoostClassifier	0.8622	0.6708	0.8072	0.7683
1NN Non-Trivial	0.8858	0.8208	0.8474	0.8659
Trivial	0.45	0.48	0.53	0.50

So, we can conclude that, when the DecisionTreeClassifier is trained with labelled points after allocating 20% of the train data as unlabelled, it gives the best accuracy. This could be because Decision Trees can handle multidimensional data and the selected Cervical Cancer dataset has 28 features. Hence, the Decision Tree Classifier model performs best with the particular dataset for Supervised Learning.

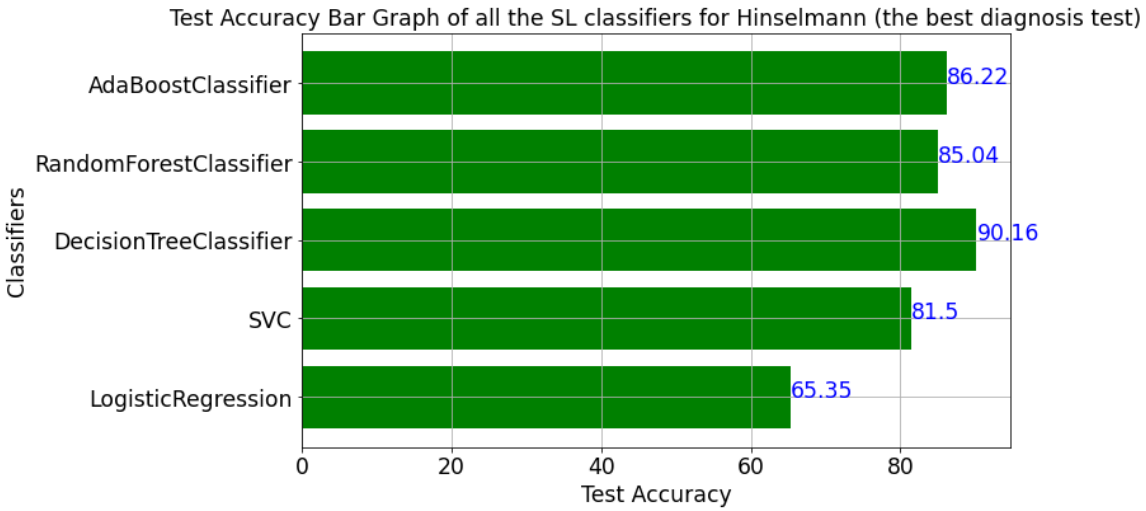

Figure 8: Accuracy comparison between different SL classifiers

Results for 20% unlabelled data:

- Best test: Hinselmann
- Best classifier: DecisionTreeClassifier with accuracy 86.22%.
- Trivial classifier: Accuracy is 45%.
- Non-Trivial classifier (1NN): Accuracy is 88.58%.
- **Therfore, DecisionTreeClassifier performs better than the baseline systems.**
- E_{out} = 1 - 0.8622 (test accuracy) = 0.137
- E_{in} = 1 - 0.9199 (train accuracy) = 0.080
- E_{out} - E_{in} = **0.051**
- D (D features) = 28
- N (training points) = 812 (407+405)
- N (testing points) = 254
- Expressing the upper bound on the out-of-sample error as $E_{out}(h_g) \leq E_{in}(h_g) + \epsilon_{vc}$. For $E_{in}(h_g)$ measures on training data, d_{vc} is used to get the value ϵ_{vc}.

  $E_{eff} \leq \sqrt{\frac{8}{N} \ln \frac{4m_h(2N)}{\delta}} = E_{vc} = \sqrt{\frac{8}{N} \ln \frac{4((2N)^{d_{vc}}+1)}{\delta}} =$ **1.465**
- Using test data to bound E_{out},

  $E_m \leq \sqrt{\frac{1}{2N} \ln \frac{2M}{\delta}} =$ **0.076**

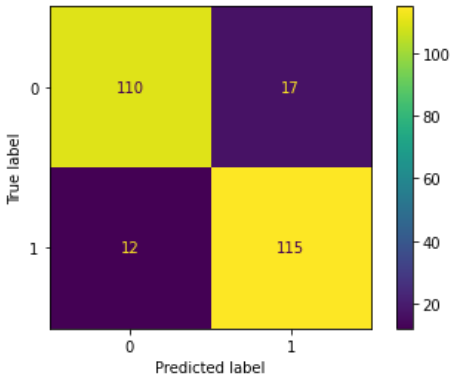

Figure 9: 1NN

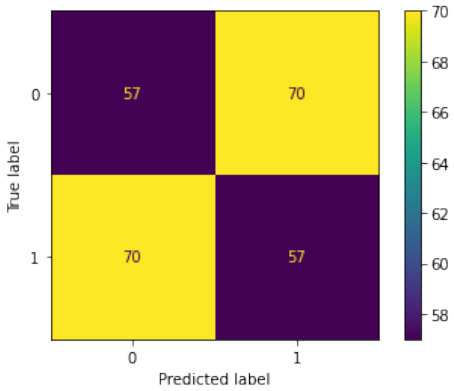

Figure 10: Trivial

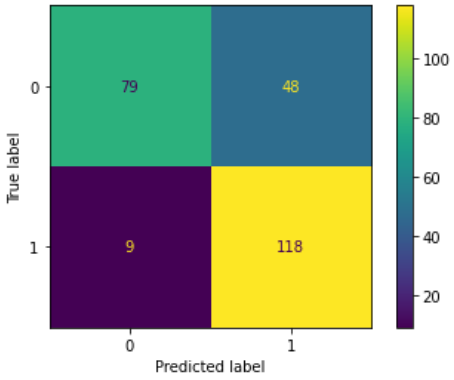

Figure 11: DecisionTreeClassifier

Figure 12: Comparing Confusion Matrices of Hinselmann Test for Non-Trivial Baseline 1NN and Trivial Baseline Random Classifier, with the best model, Decision Tree

4.2 Testing Accuracy for 60% unlabelled Data

The Table 22 provides the accuracy score report for each of the classifier against each of the tests. On finding the target class having the maximum test accuracy for each of the classifiers in the code, we obtained the result as follows:

- For KNeighborsClassifier, the best test is: Biopsy
- For LogisticRegression, the best test is: Hinselmann
- For SVC, the best test is: Hinselmann
- For DecisionTreeClassifier, the best test is: Hinselmann
- For RandomForestClassifier, the best test is: Hinselmann
- For AdaBoostClassifier, the best test is: Hinselmann

Therefore, Hinselmann test is the best test. Figure 13 shows the accuracy score for each of the classifier for Hinselmann test.

Table 22: Summary of Testing accuracy

	Hinselmann	Schiller	Citology	Biopsy
LogisticRegression	0.7087	0.6458	0.6185	0.5894
SVC	0.7835	0.7542	0.7229	0.7236
DecisionTreeClassifier	0.8268	0.6167	0.5863	0.6829
RandomForestClassifier	0.8543	0.7792	0.7711	0.7846
AdaBoostClassifier	0.7913	0.6958	0.7711	0.7033
1NN Non-Tivial	0.8622	0.80	0.8273	0.8171
Tivial	0.45	0.48	0.53	0.51

So, we can conclude that, when the Random Forest Classifier is trained with labelled points after allocating 60% of the train data as unlabelled, it gives the best accuracy. This is because the Random Forest model utilizes feature subsets selected randomly, thus performs well with high dimensional datasets, as the selected Cervical Cancer dataset. It also does not require many data transformations in preprocessing, as it does not need any assumptions about the data or its distribution.

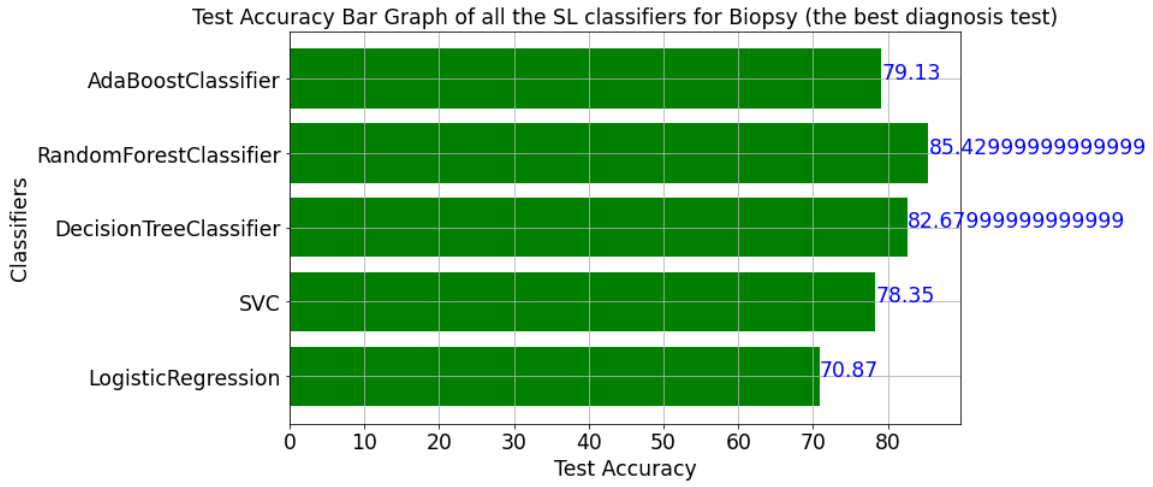

Figure 13: Accuracy comparison between different SL classifiers

Results:

- Best test: Hinselmann
- Best classifier: RandomForestClassifier with accuracy 82.67%.
- Trivial classifier: Accuracy is 45%.
- Non-Trivial classifier (1NN): Accuracy is 86.22%.
- **Therfore, RandomForestClassifier performs better than the baseline systems.**
- E_{out} = 1 - (test accuracy) = 1 - 0.8267 = 0.173
- E_{in} = 1 - (train accuracy) = 1 - 0.8473 = 0.158
- E_{out} - E_{in} = **0.015**
- D (D features) = 28
- N (training points) = 204 + 202 = 406
- N (testing points) = 254
- Expressing the upper bound on the out-of-sample error as $E_{out}(h_g) \leq E_{in}(h_g) + \epsilon_{vc}$. For $E_{in}(h_g)$ measures on training data, d_{vc} is used to get the value ϵ_{vc}.

  $E_{eff} \leq \sqrt{\frac{8}{N} \ln \frac{4m_h(2N)}{\delta}} = E_{vc} = \sqrt{\frac{8}{N} \ln \frac{4((2N)^{d_{vc}}+1)}{\delta}} = 1.975$
- On the test data,

  $E_m \leq \sqrt{\frac{1}{2N} \ln \frac{2M}{\delta}} = 0.076$

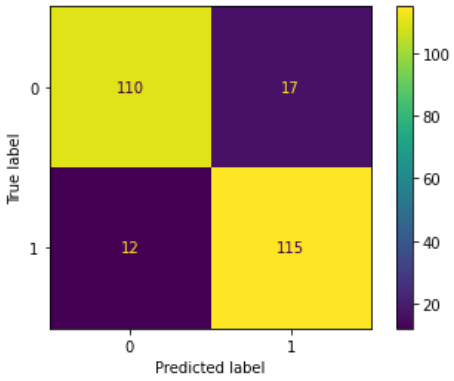

Figure 14: 1NN

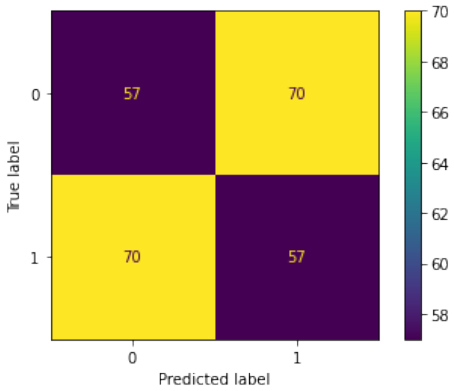

Figure 15: Trivial

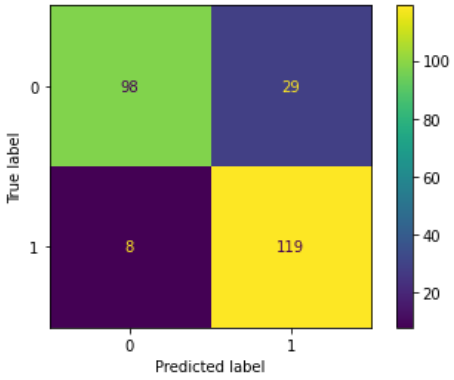

Figure 16: RandomForestClassifier

Figure 17: Comparing Confusion Matrices of Hinselmann Test for Non-Trivial Baseline 1NN and Trivial Baseline Random Classifier, with the best model, Random Forest

5 Implementation for the extension: Semi-Supervised Learning (SSL)

5.1 Data Set

The dataset used for SSL is derived from the same as used for SL, that is, the 'Cervical Cancer (Risk Factors) Data Set'. It has been downloaded from the UCI Machine Learning Dataset Repository and can be found at [3].

The original dataset has been adapted to be suitable for the SSL approach and the details of converting the SL dataset into an SSL dataset have been highlighted in section 5.2.

5.2 Dataset Methodology

- Step 1 (Train Test Split) and Step 2 (Preprocessing)

  These steps from the SL approach, in section 3.2 followed in the usage of the dataset, are the same for the SSL approach.

- Step 3 (Conversion of the SL Dataset to SSL Dataset):

  For adapting the dataset to the SSL approach, labels were removed randomly for specific percentages of data samples using the stratified random sampling technique. The train_test_split() module of sklearn.model_selection was used with its 'stratify' parameter set to 'y_train, that is, the class labels in the original train dataset. Two different percentages of unlabeled data points have been used, 20% and 60%, and these values were mentioned in the 'test_size' parameter of the train_test_split() module.

  The unlabeled targets were set to -1, thus generating three target label values, that is, 0 for cancer not detected, 1 for cancer detected, and -1 for unlabeled data points. This was done for all 4 target variables, that is, the 4 diagnosis tests for Cervical Cancer detection, namely Hinselmann, Schiller, Cytology, and Biopsy. The labeled and unlabeled features and the labeled and unlabeled target variables were concatenated to get new feature vectors and target vectors having the specified percentages of unlabeled data.

- Step 4 (Model Selection and Hyperparameter Tuning using Cross Validation):

  In SSL, both the labeled as well as the unlabeled data points are used for model selection using stratified k-fold grid search cross validation algorithm to select the model with the best hyperparameters. The number of folds(k) is set to 3.

- Step 5 (Training the Semi-Supervised Models):

  Here, the listed classifiers are trained with the best parameters using both the labelled as well as the unlabelled data points in the train data.

- Step 6 (Testing the Trained SSL Models):

  This is done on the test data to predict whether a particular patient has cancer or not, using the trained model for each of the classifiers. The best classifier selected is the one that results in the highest test accuracy score. The best diagnosis test for Cervical Cancer detection out of the 4 tests (target variables Hinselmann, Schiller, Cytology, and Biopsy) is determined as the one resulting in the highest test accuracy for the selected best classifier model.

5.3 Preprocessing and EDA

Preprocessing and EDA for Semi-Supervised Learning is the same as described in section 3.3 for Supervised Learning.

5.4 Training Process

- The same classifier models, as described in section 3.4 for Supervised Learning (SL), have been used for Semi-Supervised Learning (SSL). The difference is that in SSL, both labeled and unlabeled data has been used to train the different classifiers, as compared to SL, wherein only the labeled data points were used for training.
- The SSL approach has been implemented using the SelfTrainingClassifier of the sklearn.semi_supervised library.
- Semi-supervised learning is a situation in which in our training data, some of the samples are not labeled. The semi-supervised estimators in sklearn.semi_supervised are able to make use of this additional unlabeled data to better capture the shape of the underlying data distribution and generalize better to new samples [15].
- It is important to assign an identifier to unlabeled points along with the labeled data when training the model with the fit() method. The identifier that this implementation uses is the integer value -1 [15]. Therefore, as mentioned in Step 3 of section 5.2, in order to convert the SL dataset into an SSL dataset, the unlabeled data points have been set to -1 value for the target variables.
- The self-training implementation in sklearn.semi_supervised is based on Yarowsky's algorithm, using which a given supervised classifier can function as a semi-supervised classifier, allowing it to learn from unlabeled data [15].
- The SelfTrainingClassifier module can be called with any classifier that implements predict_proba, passed as the parameter base_classifier. In each iteration, the base_classifier predicts labels for the unlabeled samples and adds a subset of these labels to the labeled dataset [15].
- The choice of this subset is determined by the selection criterion. This selection can be done using a threshold on the prediction probabilities, or by choosing the k_best samples according to the prediction probabilities [15].
- The labels used for the final fit as well as the iteration in which each sample was labeled are available as attributes. The optional max_iter parameter specifies how many times the loop is executed at most. The max_iter parameter may be set to None, causing the algorithm to iterate until all samples have labels or no new samples are selected in that iteration [15].

5.5 Model Selection and Comparison of Results

The model selection process description and the algorithm for hyperparameter tuning are the same for SSL as explained in detail in the section 3.5. The best hyperparameters are also the same for both SL and SSL.

The only difference is that, in SSL, we are using both the unlabeled as well as the labeled data points for training, as compared to SL, wherein only the labeled data was used for training.

5.5.1 Hinselman Test

- Train data with 20% unlabeled data and the remaining labeled data is used for the SSL approach and the best hyperparameters are selected for each of the classifiers.
- Table 13 in sub-section 3.5.1 of section 3.5 lists the best hyperparameters for the Hinselman Test for 20% unlabelled data.
- Table 23 lists the training accuracy of the Self Training algorithm for the Hinselmann Test with the SSL dataset with 20% unlabeled data.

- Figure 18 depicts the decision boundary for k=2 best features of the model trained with labelled datapoints of the dataset where 20% of the data is unlabelled.
- Similar steps were followed for dataset wherein 60% of the datapoints was masked to get unlabeled points.

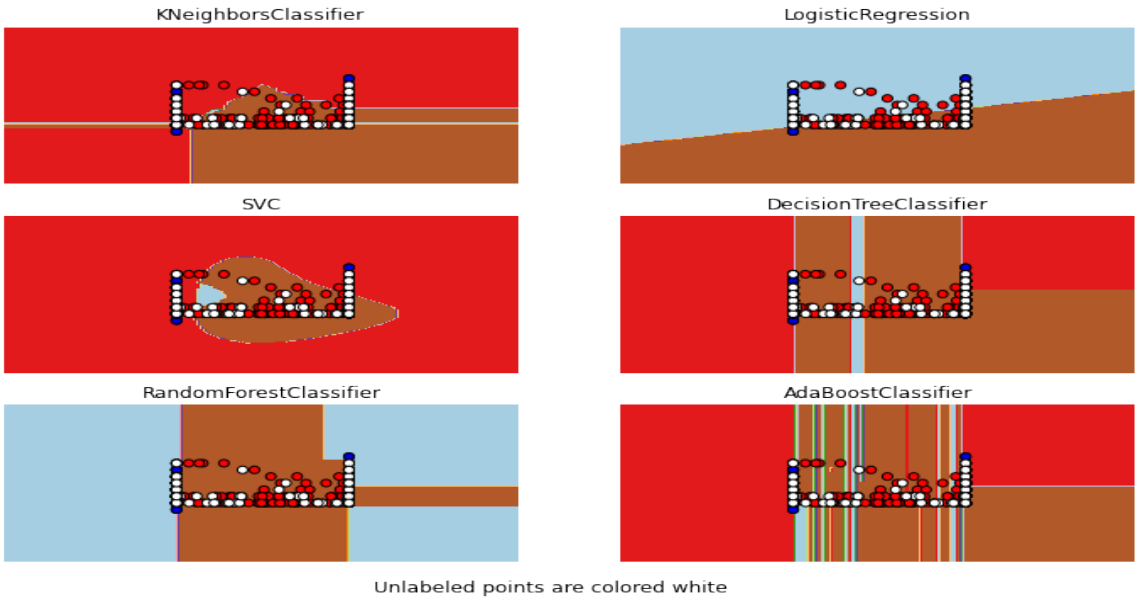

Figure 18: SSL Hinselman Test Decision Boundary on top 2 best features using KBest

Table 23: Training Accuracy of the Self Training algorithm for the Hinselmann Test with the SSL dataset with 20% unlabeled data

k folds	3
KNeighborsClassifier	Accuracy(%)
	77.73%
LogisticRegression	Accuracy(%)
	62.16%
SVC	Accuracy(%)
	72.12%
DecisionTreeClassifier	Accuracy(%)
	79.01%
RandomForestClassifier	Accuracy(%)
	74.58%
AdaBoostClassifier	Accuracy(%)
	79.70%

5.5.2 Schiller Test

- Train data with 20% unlabeled data and the remaining labeled data is used for the SSL approach and the best hyperparameters are selected for each of the classifiers.
- Table 15 in sub-section 3.5.2 of section 3.5 lists the best hyperparameters for the Schiller Test for 20% unlabelled data.
- Table 24 lists the training accuracy of the Self Training algorithm for the Schiller Test with the SSL dataset with 20% unlabeled data.
- Figure 19 depicts the decision boundary for k=2 best features of the model trained with labelled datapoints of the dataset where 20% of the data is unlabelled.
- Similar steps were followed for dataset wherein 60% of the datapoints was masked to get unlabeled points.

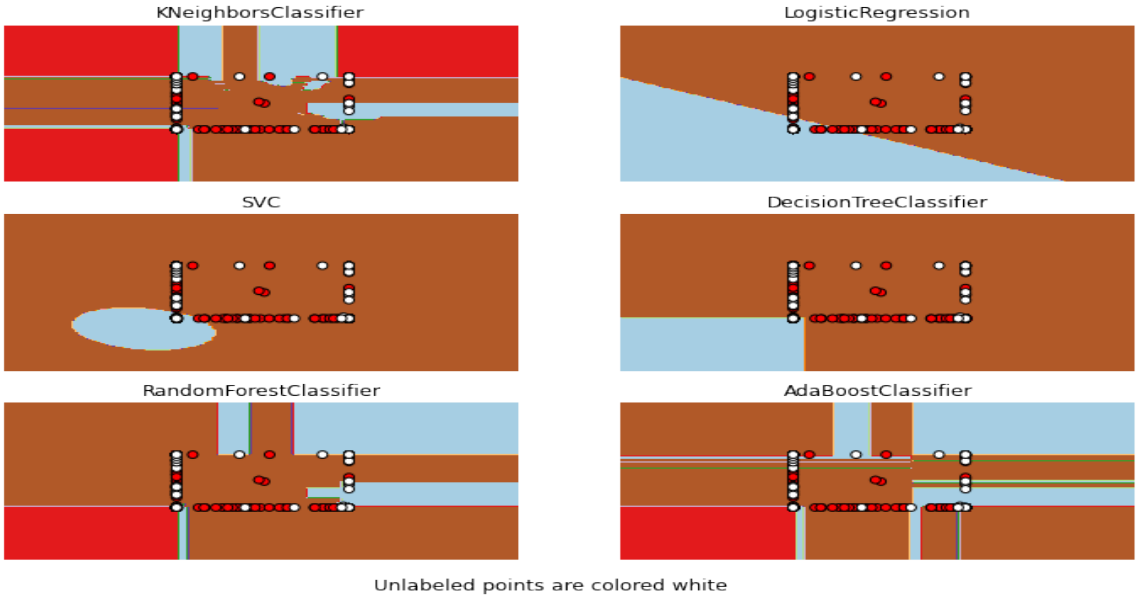

Figure 19: SSL Schiller Test Decision Boundary on top 2 best features using KBest

Table 24: Training Accuracy of the Self Training algorithm for the Schiller Test with the SSL dataset with 20% unlabeled data

k folds	3
KNeighborsClassifier	Accuracy(%)
	77.22%
LogisticRegression	Accuracy(%)
	55.17%
SVC	Accuracy(%)
	71.89%
DecisionTreeClassifier	Accuracy(%)
	77.53%
RandomForestClassifier	Accuracy(%)
	74.92%
AdaBoostClassifier	Accuracy(%)
	79.52%

5.5.3 Cytology Test

- Train data with 20% unlabeled data and the remaining labeled data is used for the SSL approach and the best hyperparameters are selected for each of the classifiers.
- Table 17 in sub-section 3.5.3 of section 3.5 lists the best hyperparameters for the Cytology Test for 20% unlabelled data.
- Table 25 lists the training accuracy of the Self Training algorithm for the Cytology Test with the SSL dataset with 20% unlabeled data.
- Figure 20 depicts the decision boundary for k=2 best features of the model trained with labelled datapoints of the dataset where 20% of the data is unlabelled.
- Similar steps were followed for dataset wherein 60% of the datapoints was masked to get unlabeled points.

Table 25: Training Accuracy of the Self Training algorithm for the Cytology Test with the SSL dataset with 20% unlabeled data

k folds	3
KNeighborsClassifier	Accuracy(%)
	76.03%
LogisticRegression	Accuracy(%)
	56.59%
SVC	Accuracy(%)
	67.77%
DecisionTreeClassifier	Accuracy(%)
	78.25%
RandomForestClassifier	Accuracy(%)
	73.51%
AdaBoostClassifier	Accuracy(%)
	78.75%

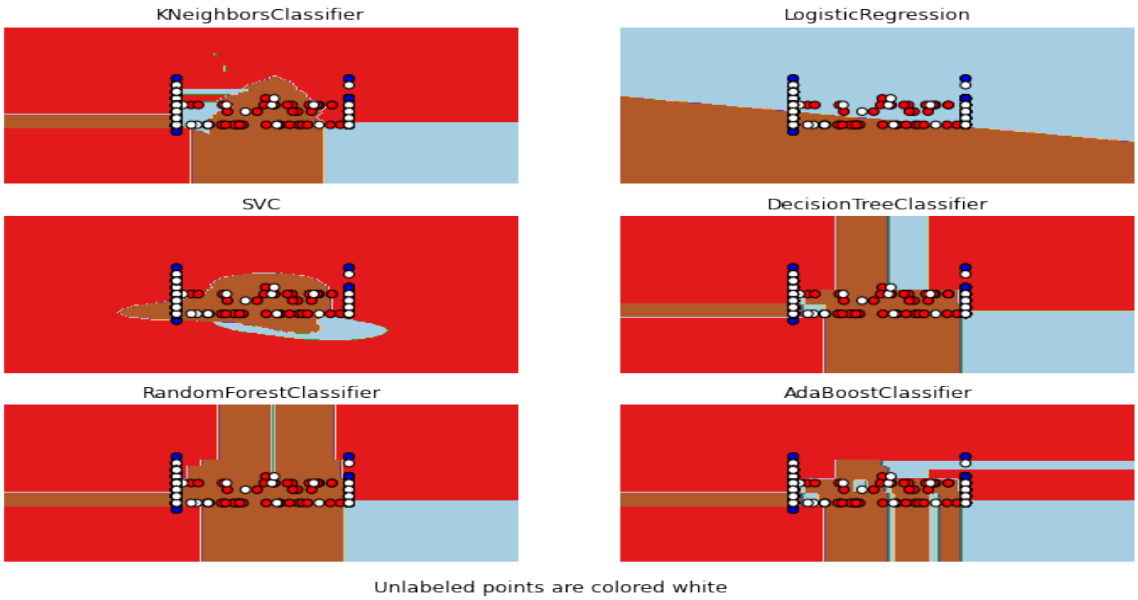

Figure 20: SSL Cytology Test Decision Boundary on top 2 best features using KBest

5.5.4 Biopsy Test

- Train data with 20% unlabeled data and the remaining labeled data is used for the SSL approach and the best hyperparameters are selected for each of the classifiers.
- Table 19 in sub-section 3.5.4 of section 3.5 lists the best hyperparameters for the Biopsy Test for 20% unlabelled data.
- Table 26 lists the training accuracy of the Self Training algorithm for the Biopsy Test with the SSL dataset with 20% unlabeled data.
- Figure 20 depicts the decision boundary for k=2 best features of the model trained with labelled datapoints of the dataset where 20% of the data is unlabelled.
- Similar steps were followed for dataset wherein 60% of the datapoints was masked to get unlabeled points.

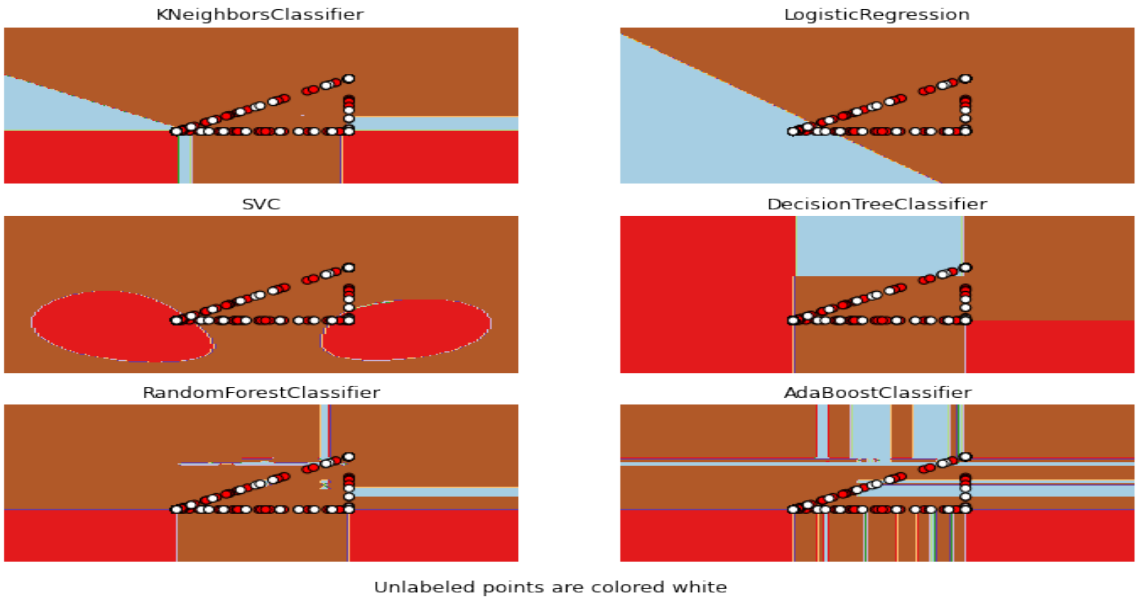

Figure 21: SSL Biopsy Test Decision Boundary on top 2 best features using KBest

Table 26: Training Accuracy of the Self Training algorithm for the Biopsy Test with the SSL dataset with 20% unlabeled data

k folds	3
KNeighborsClassifier	Accuracy(%)
	78.59%
LogisticRegression	Accuracy(%)
	52.70%
SVC	Accuracy(%)
	75.23%
DecisionTreeClassifier	Accuracy(%)
	78.89%
RandomForestClassifier	Accuracy(%)
	76.45%
AdaBoostClassifier	Accuracy(%)
	79.82%

6 Final Results and Interpretation for the extension

The testing was performed on the model trained with both labeled and unlabeled data points for the SSL approach, after partitioning 20% of the data as unlabelled and was similarly done for 60%.

The performance metric to choose the best model/classifier and the best class (the best test for Cervical Cancer detection among Hinselmann, Schiller, Cytology, or Biopsy) is the **accuracy** metric.

Steps involved are :

- Step 1: Select trained models for 20% unlabeled data.
- Step 2: For each of the classifiers (KNeighborsClassifier, LogisticRegression, SVC, DecisionTreeClassifier, RandomForestClassifier, AdaBoostClassifier), select the test which gives the highest accuracy. The highest accuracy helps us conclude that "this test" is the best for this "classifier" to predict if the patient has cancer or not based on the risk factors.

- Step 3: Once, the best test is selected, then the best classifier is selected.
- Step 4: Repeat Steps 1 to 3 for trained models for 60% unlabeled data.
- Step 5: Compare the results of models with 20% unlabeled data and 60% unlabeled data.

6.1 Testing Accuracy for 20% Unlabelled Data

The Table 27 provides the accuracy score report for each classifier model against each test. On finding the target class having the maximum test accuracy for each of the classifiers in the code, we obtained the result as follows:

- For KNeighborsClassifier, the best test is: Hinselmann
- For LogisticRegression, the best test is: Hinselmann
- For SVC, the best test is: Hinselmann
- For DecisionTreeClassifier, the best test is: Hinselmann
- For RandomForestClassifier, the best test is: Hinselmann
- For AdaBoostClassifier, the best test is: Hinselmann

Therefore, Hinselmann test is the best test. Figure 22 shows the accuracy score for each classifier for Hinselmann test.

Table 27: Summary of Testing accuracy

	Hinselmann	Schiller	Citology	Biopsy
LogisticRegression	0.6496	0.6125	0.6305	0.6057
SVC	0.8228	0.7833	0.7871	0.7276
DecisionTreeClassifier	0.7756	0.7125	0.5743	0.7317
RandomForestClassifier	0.8504	0.8	0.755	0.8333
AdaBoostClassifier	0.9055	0.6833	0.7751	0.7683
KNN Baseline	0.8622	0.8208	0.8273	0.8415

So, we can conclude that, for the best test, that is, Hinselmann test, when the AdaBoost Classifier is trained with labeled and unlabeled data points after allocating 20% of the train data as unlabelled, it gives the best accuracy. This is because the AdaBoost model has been used with Decision Tree as the base classifier and it elevates its performance by combining multiple units of the base classifier. Thus, it performs best for multiclass classification problems, as in the selected Cervical Cancer dataset, with Semi-Supervised Learning.

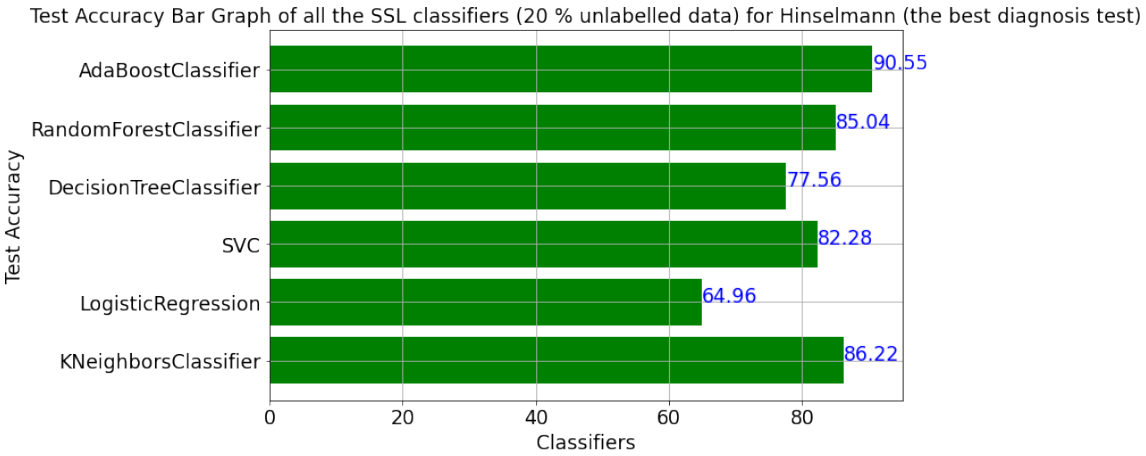

Figure 22: Accuracy comparison between different SSL classifiers

Results for 20% unlabelled data:

- Best test: Hinselmann
- Best classifier: AdaBoost Classifier with accuracy 90.55%.
- Baseline classifier (KNN): Accuracy is 86.22%.
- **Therfore, AdaBoost Classifier performs better than the baseline systems.**
- E_{out} = 1 - 0.9055 (test accuracy) = 0.0945
- The E_{out} for SL trained with 20% unlabeled data removed from the dataset was 0.1370, which is higher than the above E_{out} for SSL, that is, 0.0945, trained with 20% unlabeled data and the remaining labeled data.
- Therefore, we infer that SSL results in a better out-of-sample performance than SL by producing a lower out-of-sample error E_{out}.

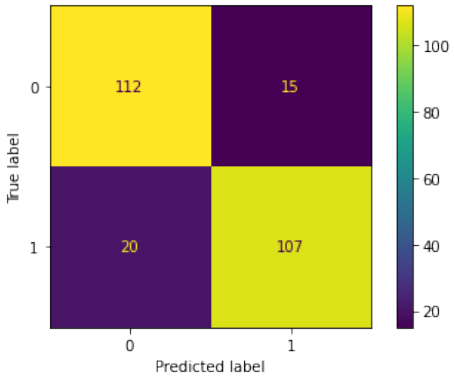

Figure 23: KNN

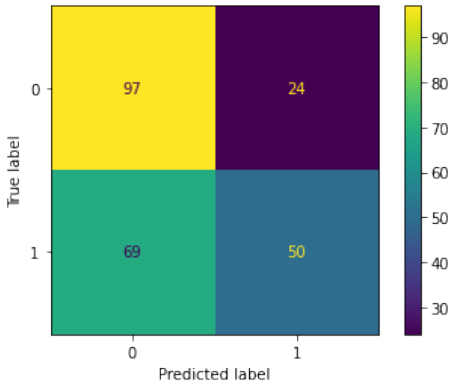

Figure 24: AdaBoost Classifier

Figure 25: Comparing Confusion Matrices of Hinselmann Test for Baseline KNN and best model AdaBoost

6.2 Testing Accuracy for 60% Unlabelled Data

The Table 28 provides the accuracy score report for each classifier model against each test. On finding the target class having the maximum test accuracy for each of the classifiers in the code, we obtained the result as follows:

- For KNeighborsClassifier, the best test is: Hinselmann
- For LogisticRegression, the best test is: Hinselmann
- For SVC, the best test is: Hinselmann
- For DecisionTreeClassifier, the best test is: Hinselmann
- For RandomForestClassifier, the best test is: Hinselmann
- For AdaBoostClassifier, the best test is: Cytology

Therefore, Hinselmann test is the best test. Figure 26 shows the accuracy score for each classifier for Hinselmann test.

Table 28: Summary of Testing accuracy

	Hinselmann	Schiller	Citology	Biopsy
LogisticRegression	0.6693	0.6292	0.6305	0.6057
SVC	0.8386	0.7833	0.7751	0.7724
DecisionTreeClassifier	0.7756	0.7	0.6024	0.6423
RandomForestClassifier	0.8543	0.7958	0.7791	0.8333
AdaBoostClassifier	0.7559	0.5958	0.8112	0.7236
KNN Baseline	0.8622	0.8208	0.8273	0.8415

So, we can conclude that, for the best test, that is, Hinselmann test, when the KNN (K Nearest Neighbors) is trained with labeled and unlabeled data points after allocating 60% of the train data as unlabelled, it gives the best accuracy. This is an unusual result because even though the KNN is a baseline classifier, it is performing better than other classifiers such as Random Forest, which is resulting in the second highest accuracy after KNN. This could be due to the fact that KNN is a clustering algorithm and hence with a large amount of unlabeled data points, that is, 60% in this case, for SSL, it performs better in clustering the unlabeled data points, and this is further improved by the presence of the remaining labeled data points in the dataset.

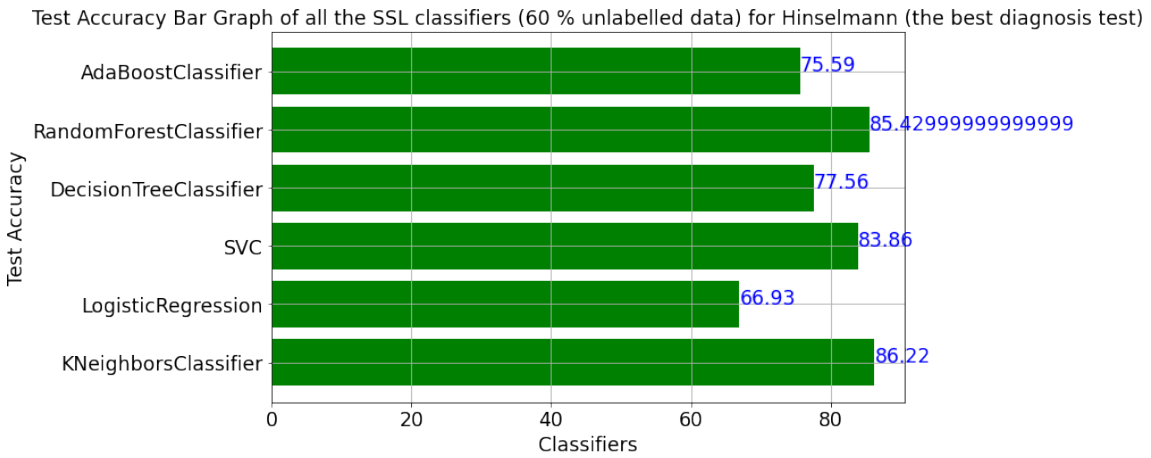

Figure 26: Accuracy comparison between different SSL classifiers

Results for 60% unlabelled data:

- Best test: Hinselmann
- Best classifier: K Nearest Neighbors with accuracy 86.22%.
- Baseline classifier (KNN): Accuracy is 86.22%.
- **The baseline classifier, KNN, performs better than the other classifier models.** A possible reason for this result has been explained in the above, after 28.
- E_{out} = 1 - 0.8622 (test accuracy) = 0.1378
- The E_{out} for SL trained with 60% unlabeled data removed from the dataset was 0.173, which is higher than the above E_{out} for SSL, that is, 0.1378, trained with 60% unlabeled data and the remaining labeled data.
- Therefore, we infer that SSL results in a better out-of-sample performance than SL by producing a lower out-of-sample error E_{out}.

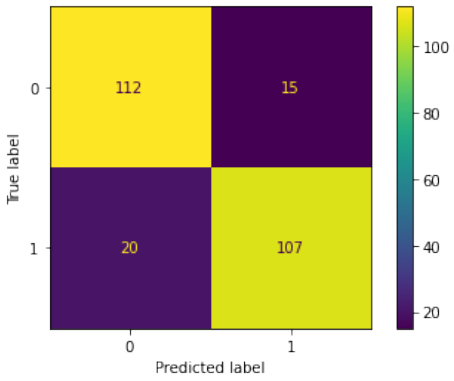

Figure 27: Confusion Matrix of Hinselmann Test for KNN

7 Contributions of each Team Member

1. Project Proposal/Planning
    - Equal contribution in dataset search and defining project goals/approach
2. Literature Survey
    - Divya: Research Paper [1]
    - Maitreyee: Research Paper [2]
3. Coding (collaborative code using Google Colab)
    - Divya:
        - EDA (50 %)
        - Preprocessing (50 %)
        - Baseline System 1: Random Probability Classifier
        - 3 SL Classifiers: Decision Tree, Random Forest, AdaBoost
        - Conversion of SL dataset to SSL (50 %)
        - 2 SSL Classifiers: Logistic Regression, SVM
    - Maitreyee:
        - EDA (50 %)
        - Preprocessing (50 %)
        - Baseline System 1: Majority/Random Classifier
        - 2 SL Classifiers: Logistic Regression, SVM
        - Conversion of SL dataset to SSL (50 %)
        - 3 SSL Classifiers: Decision Tree, Random Forest, AdaBoost
4. Result Analysis
    - Divya:
        - EDA (50 %)
        - 3 SL Classifiers vs SSL Classifiers
        - 2 SSL Classifiers vs Baseline System
    - Maitreyee:
        - EDA (50 %)
        - 2 SL Classifiers vs SSL Classifiers
        - 3 SSL Classifiers vs Baseline System
5. Project Report
    - Divya: Problem Type, Statement/Goals, SL Approach Overview, SL (Implementation, Final Results), Summary
    - Maitreyee: Abstract, Literature Review, SSL Approach Overview, SSL (Implementation, Final Results), Conclusions

8 Summary and conclusions

8.1 Summary

The objective of this project is to detect if a person has the risk of cancer using the multi-class classification of the target labels (Hinselmann, Schiller, Cytology, Biopsy) and also which out of the four diagnosis tests is the best to detect cancer using baseline, SL and SSL models. The following steps were executed:

- Raw data was processed and cleaned to evaluate the contribution of all the risk factors (features) leading to the onset of Cervical Cancer.
- Converted the SL dataset into a Semi-Supervised Learning (SSL) dataset by removing the target labels for, first 20% and then 60%, of the original labeled data.
- Performed supervised Learning (SL) techniques like Logistic Regression, SVM, Random Forest Classifier and AdaBoost.
- Performed semi-supervised Learning (SSL) using self-training algorithm with different base estimators like Logistic Regression, SVM, Random Forest Classifier and AdaBoost.
- Used Cross validation techniques like GridSearchCV with stratified sampling to select the best hyperparameters and model.
- Computed the performance metrics (f1-score, accuracy and precision) to determine which test provides the most accurate detection. Compared the training accuracy and prediction performance of SSL with that of SL for the same classifier models used for SL [SSL dataset creation and performance evaluation].

8.1.1 Supervised Learning Results for 20% unlabelled data:

- Best test: **Hinselmann**
- Best classifier: **DecisionTreeClassifier** with accuracy 86.22%.
- Trivial classifier: Accuracy is 45%.
- Non-Trivial classifier (1NN): Accuracy is 88.58%.
- **Therfore, DecisionTreeClassifier performs better than the baseline systems.**

8.1.2 Supervised Learning Results for 60% unlabelled data:

- Best test: **Hinselmann**
- Best classifier: **RandomForestClassifier** with accuracy 82.67%.
- Trivial classifier: Accuracy is 45%.
- Non-Trivial classifier (1NN): Accuracy is 86.22%.
- **Therfore, RandomForestClassifier performs better than the baseline systems.**

8.1.3 Semi-Supervised Learning Results for 20% unlabelled data:

- Best test: **Hinselmann**
- Best classifier: **AdaBoost Classifier** with accuracy 90.55%.
- Non-Trivial classifier (KNN): Accuracy is 86.22%.
- **Therfore, AdaBoost performs better than the baseline systems.**

8.1.4 Semi-Supervised Learning Results for 60% unlabelled data:

- Best test: **Hinselmann**
- Best classifier: **K Nearest Neighbors** with accuracy 86.22%.
- Non-Trivial classifier (KNN): Accuracy is 86.22%.
- **Therfore, the baseline system, K Nearest Neighbors, performs better than the other classifier models.**

8.2 Conclusions

In Supervised Learning, we observe that when 20% of the labeled data points are masked, an accuracy of 86.22% (with Decision Tree as the best model) is achieved, whereas when 60% of the labeled points are masked, an accuracy of 82.67% (with Random Forest as the best model) is achieved. Thus, we can conclude that the Supervised Learning approach performs better with more number of labeled datapoints in the dataset, or in other words, a larger dataset results in a higher accuracy of Supervised Learning.

In Semi-Supervised Learning, we observe that when 20% unlabeled data points are present in the dataset (with 80% labeled data points), an accuracy of 90.55% (with AdaBoost as the best model) is achieved, whereas when 60% unlabeled data points are present in the dataset (with 40% labeled data points), an accuracy of 86.22% (with K Nearest Neighbors as the best model) is achieved. Thus, we can conclude that the Semi-Supervised Learning approach performs better with lesser number of unlabeled data points in the dataset, or in other words, a larger number of labeled data points present in the dataset improves the performance of Semi-Supervised Learning.

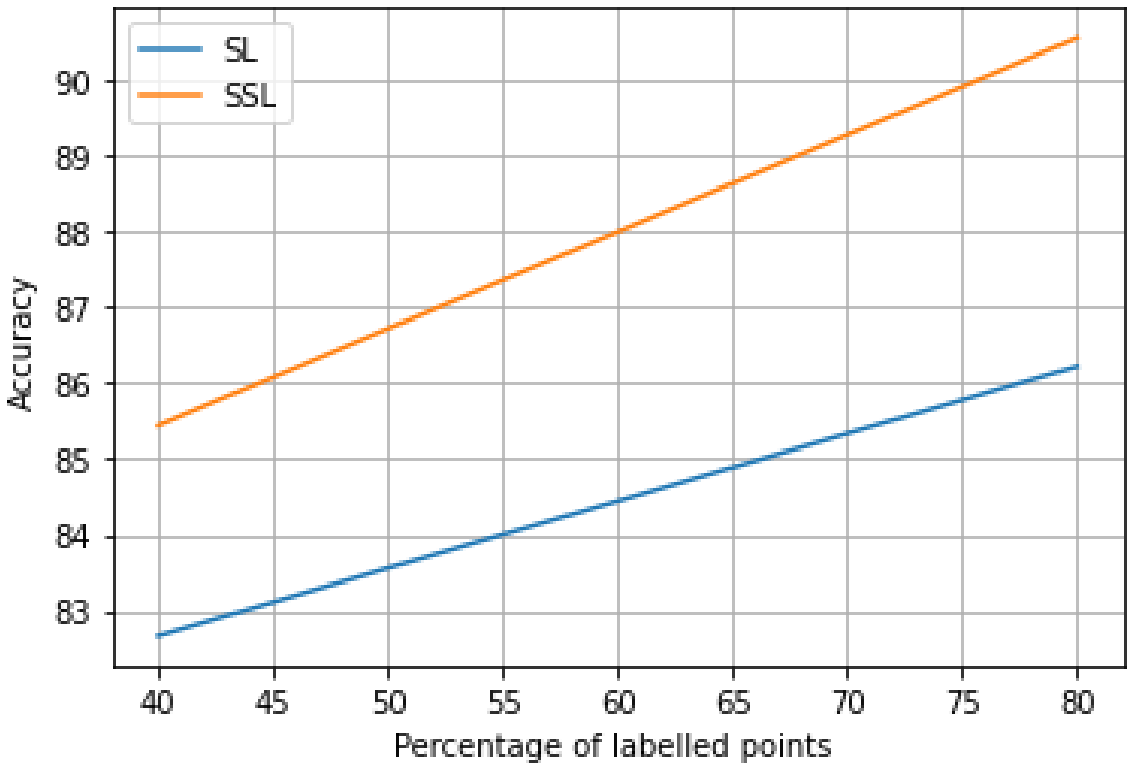

Figure 28: Conclusion

Comparing the performance of Supervised and Semi-Supervised Learning with 28, we observe that as the percentage of labeled data points increases, the accuracy of both Supervised and Semi-Supervised Learning improves, and overall, Semi-Supervised Learning results in a higher increase in accuracy than Supervised Learning, over the entire range of number of data points. Thus, we can conclude that Semi-Supervised Learning performs better than Supervised Learning for the same percentage of unlabeled data points (masked and not included for training in Supervised Learning, while included for training in Semi-Supervised Learning). This is because, with unlabeled data points used in training, the model learns to classify unseen/unknown data points (data points without labels) in the training phase itself, and thus results in a higher accuracy in the testing phase. Hence, having a mixture of labeled and unlabeled data points for training a Machine Learning model boosts its performance accuracy, as compared to having only labeled data points in the dataset.

9 References

References

[1] Zaid Alyafeai and Lahouari Ghouti, "A Fully-Automated Deep Learning Pipeline for Cervical Cancer Classification," Published in the Expert Systems With Applications Journal, September 2019. [Online]. Available: https://www.researchgate.net/publication/335937276_A_Fully-Automated_Deep_Learning_Pipeline_for_Cervical_Cancer_Classification

[2] Y. M. S. Al-Wesabi, Avishek Choudhury, and Daehan Won, "Classification of Cervical Cancer Dataset," Proceedings of the 2018 IISE Annual Conference, Binghamton University, USA, December 2018. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1812/1812.10383.pdf

[3] Dataset: K. Fernandes, J. S. Cardoso, and J. Fernandes, "Cervical Cancer (Risk Factors) Data Set," Hospital Universitario de Caracas, Caracas, Venezuela, March 2017. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Cervical+cancer+(Risk+Factors)

[4] Bernoulli Distribution based Probability Classifier (for Trivial Baseline): https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.bernoulli.html

[5] K Nearest Neighbors (for Non-Trivial Baseline): https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[6] Logistic Regression: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[7] GridSearchCV (for Model Selection): https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

[8] SVM Classifier: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[9] Decision Tree: https://towardsdatascience.com/decision-tree-classifier-explained-in-real-life-picki

[10] Decision Tree Classifier in sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

[11] Random Forest: https://www.geeksforgeeks.org/random-forest-classifier-using-scikit-learn/

[12] Random Forest Classifier in sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[13] AdaBoost: https://www.datacamp.com/tutorial/adaboost-classifier-python

[14] AdaBoost Classifier in sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

[15] SSL Self Training in sklearn: https://scikit-learn.org/stable/modules/semi_supervised.html