**Features**

1. Age Age of the patient
2. Gender Gender of the patient
3. TB Total Bilirubin
4. DB Direct Bilirubin
5. Alkphos Alkaline Phosphotase
6. Sgpt Alamine Aminotransferase
7. Sgot Aspartate Aminotransferase
8. TP Total Proteins
9. ALB Albumin
10. A/G Ratio Albumin and Globulin Ratio
11. Selector field used to split the data into two sets (labeled by the experts)

**Description**

* In the data we have total 583 liver patient records.
* This data set was collected from north east of Andhra Pradesh, India.
* Selector is a class label used to divide into groups(liver patient or not). 1 indicates that the patient is affected by liver disease and 2 indicates that the patient is not affected by liver disease

**Source:** https://archive.ics.uci.edu/ml/datasets/ILPD+(Indian+Liver+Patient+Dataset)#

**Classification**

**Input**: Age, Gender, Total Bilirubin, Direct Bilirubin, Alkaline Phosphatase, Alamine Aminotransferase ,Aspartate Aminotransferase, Total Proteins, Albumin, Albumin and Globulin Ratio

**Output**: Predict Whether a patient has a liver disease or not.

**Data-preprocessing-State:**

Before Applying Algorithms in the dataset we need to handle some of the problems like handling missing values and imbalanced data problems:

**Handling-missing-values:** In sklearn we used imputer and we use to choose “Median” value for handling missing-values and we try to check the column age and see if the dataset has age is less than zero.

**Unbalanced-Class-Problems:**

Especially when we have imbalanced classes in our datasets, some of the algorithms like Decision Trees, KNN will overfit. So we need to handle this problem and avoid imbalances.

Some techniques that we learnt in machine learning literature are

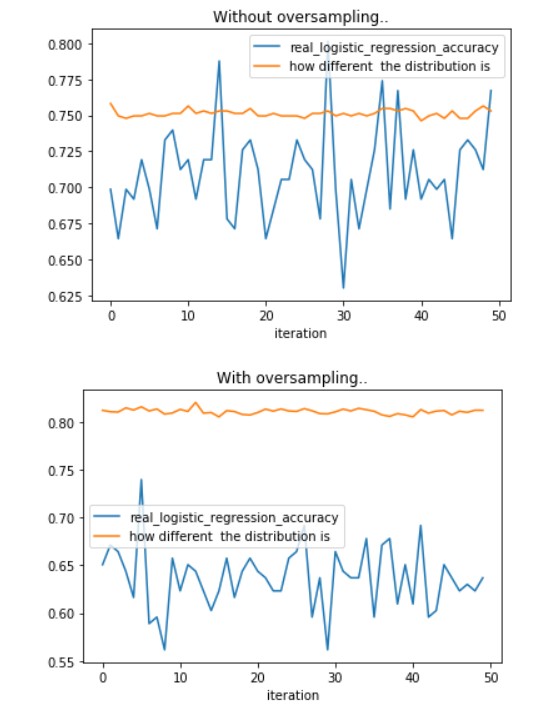
* 1.UnderSampling.
* 2.OverSampling.
* 3.Increasing Class Weights.

In this dataset we don't want to apply undersampling because, it ultimately results in data loss.

So we used oversampling techniques and there is a very good third party library to handle this. [***http://contrib.scikit-learn.org/imbalanced-learn/stable/over\_sampling.html***](http://contrib.scikit-learn.org/imbalanced-learn/stable/over_sampling.html)

We have three techniques mentioned and we tried all 3 techniques especially for algorithms where we do not have the option to apply class weights. For class weights, we used weight dictionary as {1:167,2:416} without oversampling the training data.

**Experimentation**: We see there is a huge limitation in machine learning especially for the datasets when we have training data entirely different from the testing data. In such cases mostly the accuracy goes down. This often happens in time based splitting. However we wanted to see how our accuracies vary in logistic regression based on the different random states and we present the graph down below. Here we applied stratified sampling rate.

.

**Algorithms:**

**Algorithm 1: Logistic Regression:**

We applied Logistic Regression without applying class-weights and we acquired the following results.

**Confusion-matrix:** [[94 6]  
 [39 7]]  
**Training\_score:** 0.7482837528604119  
**Testing\_score:** 0.6917808219178082

|  |  |  |  |
| --- | --- | --- | --- |
| Precision | Recall | f1-score | Support |
| 0.94 | 0.71 | 0.81 | 133 |
| 0.15 | 0.54 | 0.24 | 13 |
| 0.87 | 0.69 | 0.76 | 146 |

**Observations:**

1. Even though our classifier is predicting all minority classes wrongly we got good accuracy because of the minority class points so here we need to handle the imbalance class problems for achieving less false positive rate and increase true true positive rate.
2. Here The hyperparameter is C which is equal to 1/lambda where lambda is regularizer we applied GridSearchCv to find this parameter.

**Applying oversampling Techniques by SMOTE-Technique gives us the following results**:

LogisticRegression(C=10000, class\_weight=None, dual=False, fit\_intercept=True,  
 intercept\_scaling=1, max\_iter=100, multi\_class='ovr', n\_jobs=1,  
 penalty='l2', random\_state=None, solver='liblinear', tol=0.0001,  
 verbose=0, warm\_start=False)  
**Confusion-matrix:** [[60 40]  
 [ 9 37]]  
**Accuracy Traing\_score:** 0.7151898734177216  
**Accuracy Testing\_score:** 0.6643835616438356

|  |  |  |  |
| --- | --- | --- | --- |
| Precision | Recall | f1-score | Support |
| 0.60 | 0.87 | 0.71 | 69 |
| 0.80 | 0.48 | 0.60 | 77 |
| 0.71 | 0.66 | 0.65 | 146 |

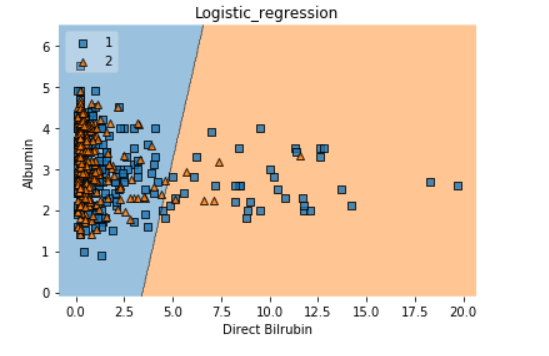
**Applying Weighted Polynomial Logistic Regression:**

LogisticRegression(C=0.001, class\_weight={1: 167, 2: 416}, dual=False,  
 fit\_intercept=True, intercept\_scaling=1, max\_iter=100,  
 multi\_class='ovr', n\_jobs=1, penalty='l2', random\_state=None,  
 solver='liblinear', tol=0.0001, verbose=0, warm\_start=False)  
**Accuracy of the model from Grid search:** 0.6164383561643836 :  
**Confusion-matrix:** [[57 10]  
 [46 33]]

|  |  |  |  |
| --- | --- | --- | --- |
| Precision | Recall | f1-score | Support |
| 0.55 | 0.85 | 0.67 | 67 |
| 0.77 | 0.42 | 0.54 | 79 |
| 0.67 | 0.62 | 0.60 | 146 |

Our main aim to get the right model which correctly balances to find the class 1 and class 2 points correctly.

This will give a rough estimate on decision boundary of logistic regression.



**Algorithm 2: Decision Tree(The crucial Hyperparameter is to find the right depth of the tree):**

Here in decision tree we applied class weight to see our algorithm performs

DecisionTreeClassifier(class\_weight={1: 167, 2: 417}, criterion='gini',  
 max\_depth=None, max\_features=None, max\_leaf\_nodes=None,  
 min\_impurity\_decrease=0.0, min\_impurity\_split=None,  
 min\_samples\_leaf=1, min\_samples\_split=2,  
 min\_weight\_fraction\_leaf=0.0, presort=False, random\_state=None,  
 splitter='best')  
**Training\_Score:** 1.0  
**Testing\_Score:** 0.7397260273972602  
**Confusion-matrix:** ([[85, 15],  
 [23, 23]])

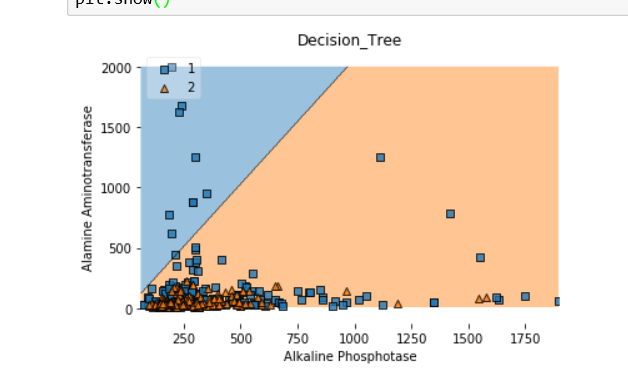
We applied 10 fold cross validation grid search Cv to find the correct depth based on cross validation Accuracy.We got the max depth =9 .

DecisionTreeClassifier(class\_weight={1: 167, 2: 417}, criterion='gini',  
 max\_depth=9, max\_features=None, max\_leaf\_nodes=None,  
 min\_impurity\_decrease=0.0, min\_impurity\_split=None,  
 min\_samples\_leaf=1, min\_samples\_split=2,  
 min\_weight\_fraction\_leaf=0.0, presort=False, random\_state=None,  
 splitter='best')  
**Training\_accuracy:** 0.88558352402746  
**Testing\_accuracy:** 0.6643835616438356  
**Confusion-matrix:** [[74 23]  
 [26 23]]

|  |  |  |  |
| --- | --- | --- | --- |
| Precision | Recall | f1-score | Support |
| 0.74 | 0.76 | 0.75 | 97 |
| 0.50 | 0.47 | 0.48 | 49 |
| 0.66 | 0.66 | 0.66 | 146 |

Controlling the depth gives good results for minority class points.

**Observations:** Decision Tree is nothing but generating lot of axis parallel lines to classify our data points which has the ability form complex decision boundary by lot of axis parallel lines.



**Algorithm 3: K-Nearest neighbor(Choosing right k is the hyperparameter)**

**Note:** K-Nearest Neighbor Classifier in sklearn do not have the option to provide the class weights however we can change the weights based on uniform and distance so we try to show the differences how our algorithm perform with and without oversampling.

Without Oversampling we applied simple cross validation to find right K:

Cross-validation Accuracy for my k=1 is 63.63636363636363  
Cross-validation Accuracy for my k=3 is 66.66666666666666  
Cross-validation Accuracy for my k=5 is 62.121212121212125  
Cross-validation Accuracy for my k=7 is 59.84848484848485  
Cross-validation Accuracy for my k=9 is 61.36363636363637  
Cross-validation Accuracy for my k=11 is 65.15151515151516  
Cross-validation Accuracy for my k=13 is 63.63636363636363  
Cross-validation Accuracy for my k=15 is 65.15151515151516  
Cross-validation Accuracy for my k=17 is 68.93939393939394  
We choose K=17 and got the confusion matrix as below:  
**Confusion-matrix:** ([[88, 40],  
 [12, 6]], )

**Applying Naive Random Oversampling Technique:**

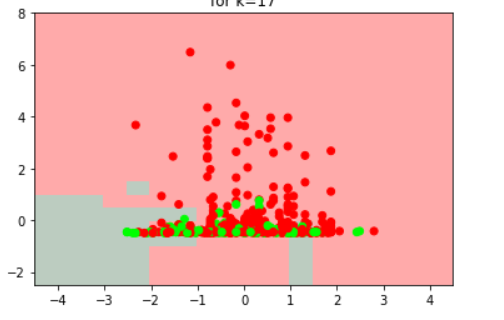
Cross-validation Accuracy for my k=1 is 89.24050632911393  
Cross-validation Accuracy for my k=3 is 74.0506329113924  
Cross-validation Accuracy for my k=5 is 72.15189873417721  
Cross-validation Accuracy for my k=7 is 78.48101265822784  
Cross-validation Accuracy for my k=9 is 75.31645569620254  
Cross-validation Accuracy for my k=11 is 74.0506329113924  
Cross-validation Accuracy for my k=13 is 75.31645569620254  
Cross-validation Accuracy for my k=15 is 74.0506329113924  
Cross-validation Accuracy for my k=17 is 77.84810126582279  
Cross-validation Accuracy for my k=19 is 75.31645569620254  
Cross-validation Accuracy for my k=21 is 78.48101265822784  
Cross-validation Accuracy for my k=23 is 74.68354430379746  
Cross-validation Accuracy for my k=25 is 74.68354430379746  
Cross-validation Accuracy for my k=27 is 74.68354430379746  
Cross-validation Accuracy for my k=29 is 75.31645569620254  
The Test Accuracy for K=1 is 60.95890410958904  
**Confusion-matrix:**  [[70 27]  
 [30 19]]

|  |  |  |  |
| --- | --- | --- | --- |
| Precision | Recall | f1-score | Support |
| 0.70 | 0.72 | 0.71 | 97 |
| 0.41 | 0.39 | 0.40 | 49 |
| 0.60 | 0.61 | 0.61 | 146 |

**Observations:**

* We got significantly better accuracy here in Knn after applying oversampling so oversampling is very helpful where we do not have the option of class\_weights to enter.
* KNN has the ability to design more complex decision Boundaries so it is a great algorithm when our data is non linear or the data which cannot be linearly separated by line or hyperplane.

Decision Boundary for KNN: for k=17



**Algorithm 4:Random-Forests(Bagging)**

We applied this Technique especially when we have high variance and low bias models

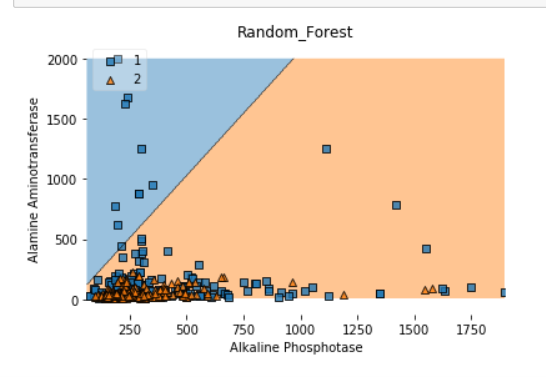
**Following results acquired without choosing right estimators in the algorithm.**

RandomForestClassifier(bootstrap=True, class\_weight={1: 167, 2: 416},  
 criterion='gini', max\_depth=None, max\_features='auto',  
 max\_leaf\_nodes=None, min\_impurity\_decrease=0.0,  
 min\_impurity\_split=None, min\_samples\_leaf=1,  
 min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,  
 n\_estimators=10, n\_jobs=1, oob\_score=False, random\_state=0,  
 verbose=0, warm\_start=False)  
**Training\_Score:** 0.977116704805492  
**Testing\_Score:** 0.6438356164383562  
**Confusion Matrix:** [[88 12]  
 [40 6]]

Choosing the number of decision Trees is the crucial parameter so we applied 10 fold cross validation using gridsearchCV.After finding the right no of estimators we see there is improvement in accuracy than above:

RandomForestClassifier(bootstrap=True, class\_weight={1: 167, 2: 416},  
 criterion='gini', max\_depth=None, max\_features='auto',  
 max\_leaf\_nodes=None, min\_impurity\_decrease=0.0,  
 min\_impurity\_split=None, min\_samples\_leaf=1,  
 min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,  
 n\_estimators=14, n\_jobs=-1, oob\_score=False, random\_state=0,  
 verbose=0, warm\_start=False)  
**Training\_accuracy:** 0.9908466819221968  
**Testing\_accuracy:** 0.6438356164383562  
**Confusion Matrix:** [[86 14]  
 [38 8]]  
**Observations:**

* Random Forest is a combination of multiple decision Trees trained on bootstrapping data techniques i.e row sampling and column sampling
* Random Forest can significantly improve algorithm accuracy when we have models of high variance and low bias and also has the ability to form complex decision Boundaries.



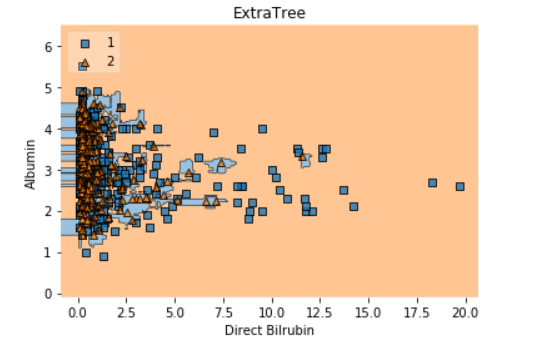
**Algorithm 5: Extremely Randomized Trees**

Extremely Randomized Trees is also a technique in bagging which has more randomness than generic random forests it has the ability to outperform Random Forests especially when our random forests overfit and perform poorly on Testing data.

Extra-Trees really perform better than Random Forests and we see it improves our accuracy when compared with Random Forests.Choosing the right no of estimators is the hyperparameter in the extra trees we applied 10 fold cross validation to find the right no of estimators and we got the results below:

No of estimators: 10

ExtraTreesClassifier(bootstrap=True, class\_weight={1: 167, 2: 416},  
 criterion='gini', max\_depth=None, max\_features='auto',  
 max\_leaf\_nodes=None, min\_impurity\_decrease=0.0,  
 min\_impurity\_split=None, min\_samples\_leaf=1,  
 min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,  
 n\_estimators=10, n\_jobs=-1, oob\_score=False, random\_state=0,  
 verbose=0, warm\_start=False)  
**Training\_accuracy:** 0.9839816933638444  
**Testing accuarcy**: 0.7054794520547946  
**Confusion matrix**: [[94 6]  
 [37 9]]



**Algorithm 6: Gradient Boosting Decision trees(Boosting)**

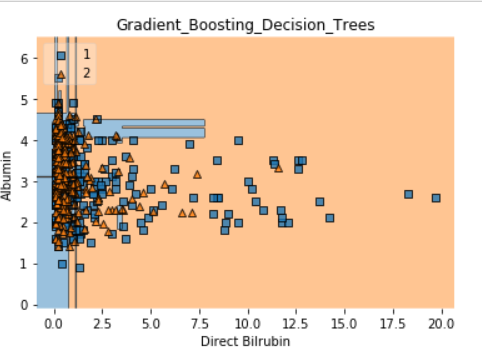
Boosting trees perform much better than Random Forests.Even though we got more accuracy in Random Forests GBDT trees were able to classify the minority class points correctly.GBDT trees will perform better when we have high bias and low variance models.

Choosing the right hyperparameters is crucial for the best algorithm.i.e maxdepth,no of estimators, trees in boosting, subsample .However it takes more time to train once the training phase is completed it is faster because the decision trees depth is equivalent to decision stump i.e every depth of the tree not more than 3

We got the following results:

GradientBoostingClassifier(criterion='friedman\_mse', init=None,  
 learning\_rate=0.1, loss='deviance', max\_depth=9,  
 max\_features=None, max\_leaf\_nodes=None,  
 min\_impurity\_decrease=0.0, min\_impurity\_split=None,  
 min\_samples\_leaf=1, min\_samples\_split=2,  
 min\_weight\_fraction\_leaf=0.0, n\_estimators=100,  
 presort='auto', random\_state=0, subsample=1.0, verbose=0,  
 warm\_start=False)  
**Training\_accuracy**:1.0  
**Testing\_accuracy:** 0.6712328767123288  
**Confusion Matrix:** [[74 26]  
 [22 24]]

|  |  |  |  |
| --- | --- | --- | --- |
| Precision | Recall | f1-score | Support |
| 0.77 | 0.74 | 0.76 | 100 |
| 0.48 | 0.52 | 0.50 | 46 |
| 0.68 | 0.67 | 0.67 | 146 |

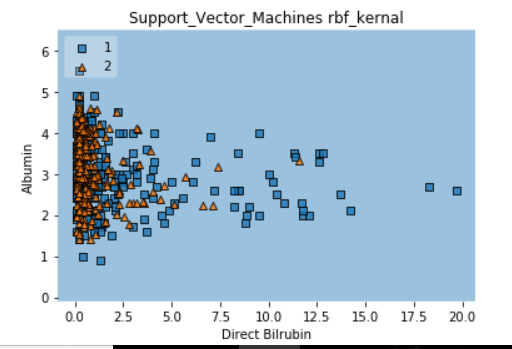


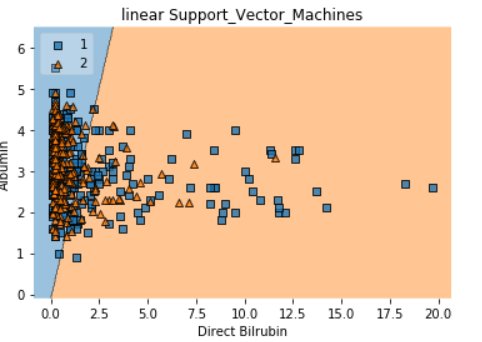
**Algorithm 7: Support Vector Machines:(RBF Kernel SVM)**

Here the hyperparameters are c i.e hinge loss we can see below that our algorithm falls underfit.We applied 10 Fold cross validation but did not get good results here.Rbf kernal svm falls in under fit and linear svm did not work well because it didnot have the ability to seperate non linear data.

SVC(C=1e-08, cache\_size=200, class\_weight={1: 167, 2: 416}, coef0=0.0,  
 decision\_function\_shape='ovr', degree=3, gamma='auto', kernel='rbf',  
 max\_iter=-1, probability=False, random\_state=0, shrinking=True,  
 tol=0.001, verbose=False)  
[[100 0]  
 [ 46 0]]  
**Accuracy for Traing\_score:** 0.7231121281464531  
**Accuracy for Testing\_score:** 0.684931506849315

|  |  |  |  |
| --- | --- | --- | --- |
| Precision | Recall | f1-score | Support |
| 1.00 | 0.68 | 0.81 | 146 |
| 0.00 | 0.00 | 0.00 | 0 |
| 1.00 | 0.68 | 0.81 | 146 |

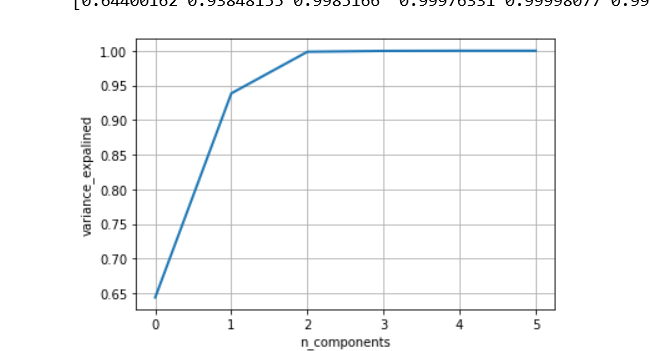
****

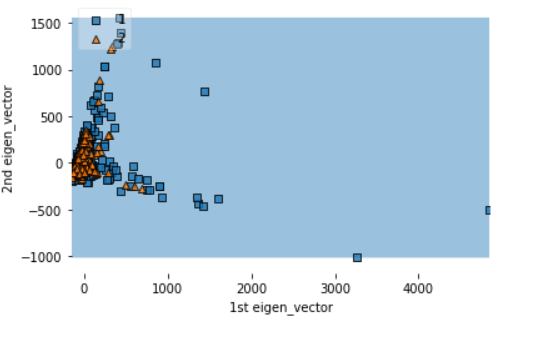
****

**Algorithm 8: PCA and Tsne**

We applied PCA and Tsne for data visualizations for seeing the higher dimensions data to lower dimension data.one disadvantage we see is in pca is this algorithm cannot preserve the local neighborhood so T stochastic Neighbor-hood Embedding has the ability to preserve the local neighborhood we applied this techniques to visualize the data.

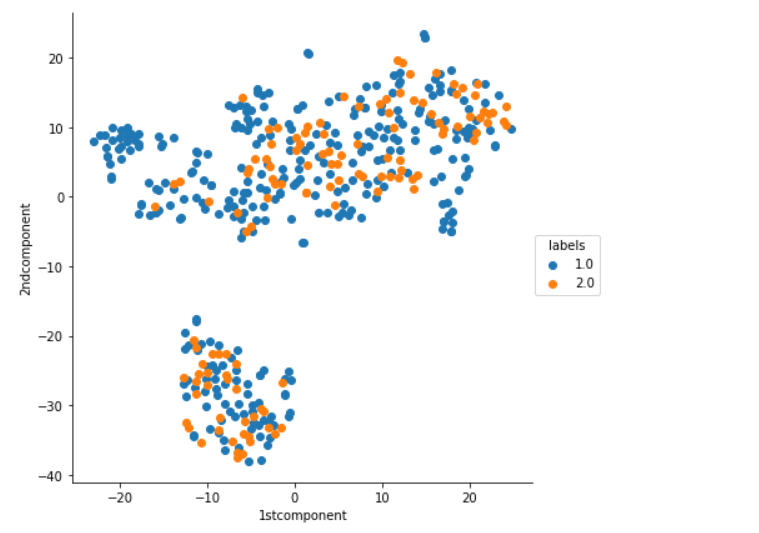
PCA: Visualization of top two eigen-vectors and plot on variance preservation vs eigen values:





**Visualization of Tsne**:

We got a rough estimate that there is some neighborhood between the points but this local neighborhood cannot be preserved in Principal component analysis.Basically here the hyperparameters are perplexity and no of iterations here we need to set the right perplexity and iterations to see how data distribution changes for different perplexities.This experimentation is just helpful for data visualization and for a rough estimate of how our n- dimensional data in 2 dimensional space.(here we choose perplexity =30 and no of iterations =1000).



**Algorithm 9: Gaussian Naive Bayes**

Gaussian Naive bayes did not work well on our dataset because naive bayes has a strong assumption of conditional independence so i think our features do not have this conditional independence and underlying distribution is not gaussian so algorithm did not perform well on our data.Similarly multinomial gaussian Naive bayes also didnot perform well but we understand this naive bayes works well on text processing where the conditional independence works well but for our data gaussian Naive Bayes did not work well.

Training\_score: 0.6116207951070336  
Testing\_score:0.5342465753424658  
[[32 0]  
 [68 46]]

**Future Scope:**

We read about stacking concept to join all the different classifier to form one meta classifier and we believe that we can expect good results by applying this technique.