Project Name
Exploring Semi-Supervised learning on California Housing Dataset

Student Name: *Tejas Mhadgut*

RIT Email ID: tm3886@rit.edu

Course: DSCI 633: Foundations of Data Science
Instructor: Dr. Nidhi Rastogi
Date: 15 December 2023

I. Data Science Problem

The focus of this project revolves around the exploration of self-training, which stands as an exemplar technique in semi-supervised learning. This exploration is executed using the California Housing Dataset.

The primary goals of this project encompass the following key objectives:

1. Transform the dataset, originally structured as a regression problem, into a classification problem.
2. Remove a specified number of labels from the dataset, creating a scenario conducive to showcasing semi-supervised techniques effectively.
3. Execute semi-supervised learning techniques using diverse models and meticulously evaluate their performances. The objective here is to ascertain the model that best aligns with the given data and problem context.

These objectives will be explained in detail further in the report.

II. Dataset and Data Exploration

Let's Explore the California housing dataset,

- This dataset was obtained from the StatLib repository.
  https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html
  We will be importing this dataset using sklearn library.
- The Characteristics of this dataset are as follows:
  1. Number of Instances: 20640
  2. Number of Attributes: 8 numeric, predictive attributes and the target
  3. Attribute Information:

- MedInc	median income in block group
- HouseAge	median house age in block group
- AveRooms	average number of rooms per household
- AveBedrms	average number of bedrooms per household
- Population	block group population
- AveOccup	average number of household members
- Latitude	block group latitude
- Longitude	block group longitude

  4. Missing Attribute Values: None
- The target variable is the median house value for California districts, expressed in hundreds of thousands of dollars ($100,000).
- This dataset was derived from the 1990 U.S. census, using one row per census block group. A block group is the smallest geographical unit for which the U.S. Census Bureau publishes sample data (a block group typically has a population of 600 to 3,000 people).

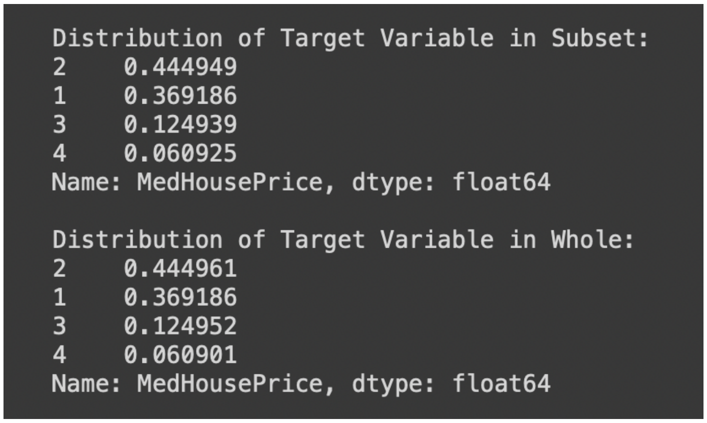
The target variable, median house value, is originally a continuous variable, indicating that the problem at hand is fundamentally a regression problem. However, the approach here involves converting this target variable into a discrete categorical variable. The rationale behind this conversion is as follows:

- Classification labels are often propagated more easily across similar instances in semi-supervised learning. Discrete labels might propagate more consistently and effectively in certain semi-supervised algorithms compared to continuous predictions.
- Classification problems can sometimes lead to simpler decision boundaries compared to regression problems. In semi-supervised learning, simpler decision boundaries can be advantageous as they may generalize better to unlabeled data.
- Certain semi-supervised learning algorithms or techniques might be more naturally suited for classification tasks. Transforming the problem into a classification format can make it easier to apply specific algorithms designed or adapted for semi-supervised learning.
- The discrete nature of class labels might sometimes offer more stable and interpretable predictions in semi-supervised settings, especially when using techniques like self-training or co-training.
- If there's a lack of labeled data points for continuous target values, leveraging unlabeled data might be less straightforward compared to classification tasks.

I'm interested in exploring the semi-supervised learning approach because it often proves to be more practical in real-world scenarios compared to supervised learning.

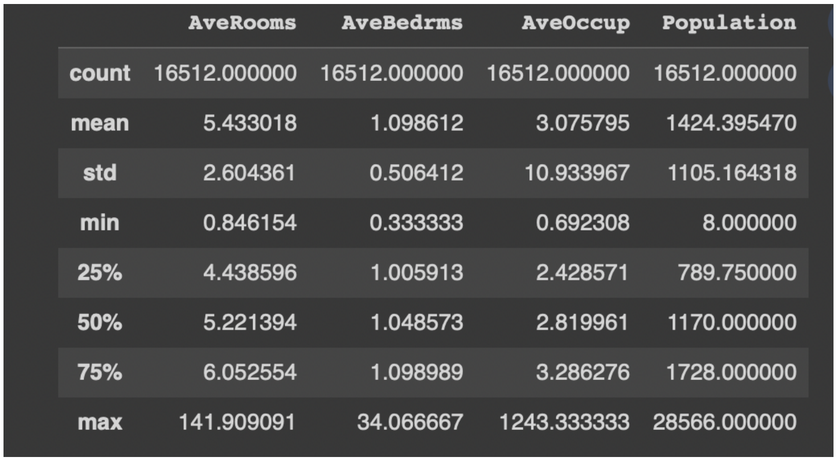
- In many domains, obtaining labeled data can be challenging due to factors like privacy concerns, expert knowledge requirements, or rare events. Semi-supervised learning makes the most out of limited labeled data by utilizing additional information from the larger pool of unlabeled data.
- Semi-supervised learning models can be updated or expanded more easily with new unlabeled data, enabling continuous learning and scalability in dynamic real-world environments.

To achieve all of our objectives,
We discretized the MedHousePrice target variable into four bins: 0.0, 1.5, 3.0, and 4.5, labeled respectively as 1, 2, 3, and 4. This transformation effectively converted the problem into a classification task.
Additionally, to ensure proportional representation of all classes in the training dataset similar to the entire dataset, we divided the data into training and testing subsets using StratifiedShuffleSplit.



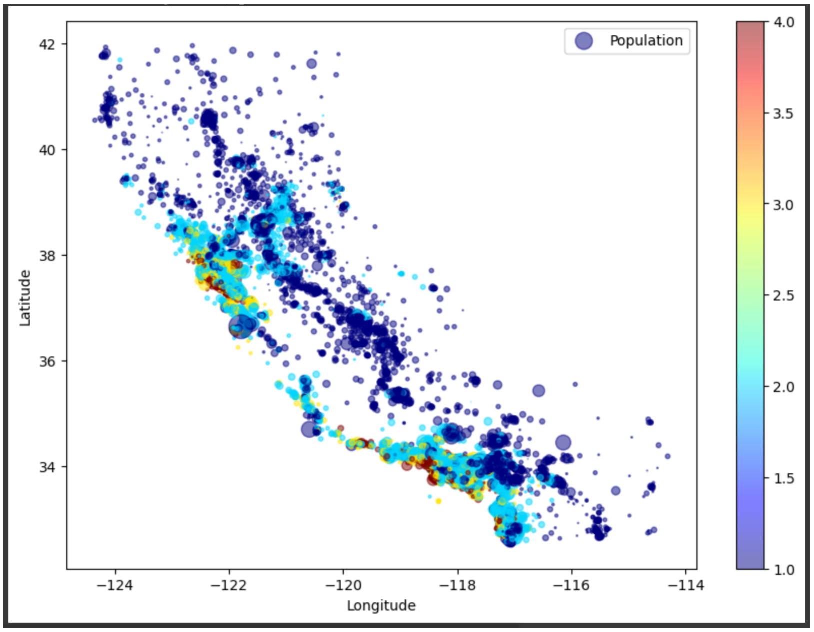
```
Distribution of Target Variable in Subset:
2    0.444949
1    0.369186
3    0.124939
4    0.060925
Name: MedHousePrice, dtype: float64

Distribution of Target Variable in Whole:
2    0.444961
1    0.369186
3    0.124952
4    0.060901
Name: MedHousePrice, dtype: float64
```

Now we perform exploratory data Analysis on the dataset



	AveRooms	AveBedrms	AveOccup	Population
count	16512.000000	16512.000000	16512.000000	16512.000000
mean	5.433018	1.098612	3.075795	1424.395470
std	2.604361	0.506412	10.933967	1105.164318
min	0.846154	0.333333	0.692308	8.000000
25%	4.438596	1.005913	2.428571	789.750000
50%	5.221394	1.048573	2.819961	1170.000000
75%	6.052554	1.098989	3.286276	1728.000000
max	141.909091	34.066667	1243.333333	28566.000000

The depicted figure highlights a significant disparity between the 75th percentile and maximum values in the mentioned fields. While these variations could typically signify outliers, in the context of this dataset, it's important to note that having data points with notably high values for Average Rooms, Average Bedrooms, Average Occupation, and Average Population can be uncommon but still valid. Considering the nature of the California housing dataset, it's plausible to encounter instances with a minority of Wealthy individuals, which could explain these observations.



Above is the scatter plot of data points. The dark red points represent people living closer to the ocean proximity, hence having most expensive housing.

III. Model implementations

Following the train-test split and exploratory data analysis, we delve into model implementation. To standardize our input data across models, we employ Standard Scaling on both the training and testing datasets.

Standard Scaling is pivotal in ensuring a level playing field for all features. By bringing them to a comparable scale, it prevents any single feature from disproportionately influencing the training process solely based on its magnitude. Instead, it ensures that each feature contributes proportionately, fostering a balanced training phase.

Next, a strategic modification is applied to the y_train variable by replacing 70% of its labels with -1. This tailored adjustment is a cornerstone of our approach, particularly when utilizing the SelfTrainingClassifier from the sklearn library within the realm of semi-supervised learning.

The SelfTrainingClassifier serves as a specialized tool in semi-supervised learning. It initiates by training a classifier on a limited labeled dataset, thereafter moving iteratively. At each iteration, it confidently predicts labels for unlabeled data points, treating them as pseudo-labeled instances. These pseudo-labeled data points are gradually integrated into the labeled set, refining predictions and potentially elevating the model's performance in subsequent iterations. This iterative mechanism harnesses the power of unlabeled data, progressively enhancing the overall model performance.

```
# Trains logistic model for semi-supervised problem
self_training_clf = SelfTrainingClassifier(logistic)
self_training_clf.fit(X_train_scaled, y_labeled)
```

The models used are as follows:

1. Logistic Regression
2. SVM
3. Random Forest
4. Decision Tree

Logistic Regression:

- Logistic Regression is a simple yet effective classification algorithm that serves as a good .starting point or baseline model in many machine learning projects.
- When dealing with limited labeled data, Logistic Regression can still perform reasonably well.
- We Put pentalty as l1 and l2 to see how model performs under regularization
- Performed GridSearchCV for hyperparameter tuning. Below are the various hyperparameters used.

  ```
  {'C': [0.1, 1, 10, 100],'max_iter': [3000, 4000, 5000]}
  ```
- We performed l1and l2 Logistic regression with and without PCA.

SVM:

- SVMs can use different kernel functions to handle complex relationships between features. The code explores SVMs with linear and RBF kernel functions to understand their impact on classification performance.
- SVMs aim to maximize the margin between classes, leading to better generalization and potentially lower generalization error on unseen data.
- For this project hyperparameter tuning is out of scope as it took extensive amount of time.
- Performed SVM both before and after PCA.

Random Forest:

- Random Forests tend to be less prone to overfitting, especially when compared to single decision trees. By aggregating predictions from multiple trees, Random Forests can generalize well to new, unseen data.
- Performed GridSearchCV for hyperparameter tuning. Below are the various hyperparameters used.

```
param_grid = {'n_estimators': [50, 100, 150],'max_depth': [None, 5, 10, 15]}
```

- Performed Random Forest before and after PCA.

Decision Tree:

- Decision Trees can model non-linear relationships between features and the target variable. They can segment the feature space into regions, making them effective for complex classification tasks.
- Similar to other classifiers like Logistic Regression, SVMs, and Random Forests, Decision Trees were used within the SelfTrainingClassifier to explore their behavior in semi-supervised learning. Decision Trees are simple and suitable candidates to observe how they adapt with additional pseudo-labeled data during the self-training process.
- Performed GridSearchCV for hyperparameter tuning. Below are the various hyperparameters used.

```
param_grid = {'max_depth':[3,5,7,10,15]        'min_samples_leaf':[3,5,10,15,20],
'min_samples_split':[8,10,12,18,20,16],
'criterion':['gini','entropy']}
```

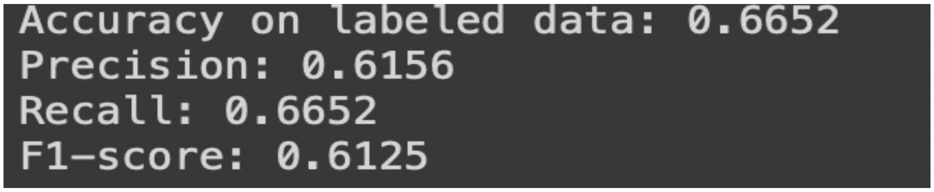
- Performed Random Forest before and after PCA.

IV. Model Evaluation

Precision: Out of all the positive predictions we made, how many were true?
Recall: Out of all the data points that should be predicted as true, how many did we correctly predict as true?
F1 Score: F1 Score is a measure that combines recall and precision. As we have seen there is a trade-off between precision and recall, F1 can therefore be used to measure how effectively our models make that trade-off.
Confusion metrics: It presents True positive, True Negative, False Positive and False Negative classifications for each class in the dataset.

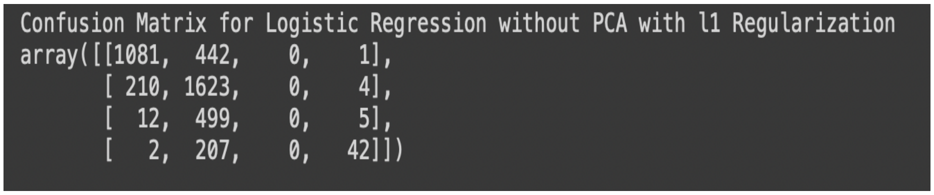
Logistic Regression l1 without PCA results:

Evaluation:



```
Accuracy on labeled data: 0.6652
Precision: 0.6156
Recall: 0.6652
F1-score: 0.6125
```

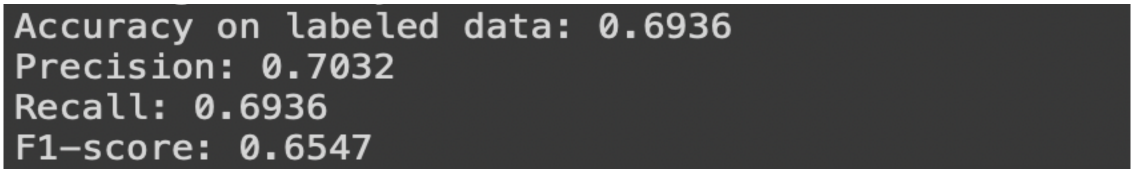
Confusion Metrics:



```
Confusion Matrix for Logistic Regression without PCA with l1 Regularization
array([[1081,  442,    0,    1],
       [ 210, 1623,    0,    4],
       [  12,  499,    0,    5],
       [   2,  207,    0,   42]])
```

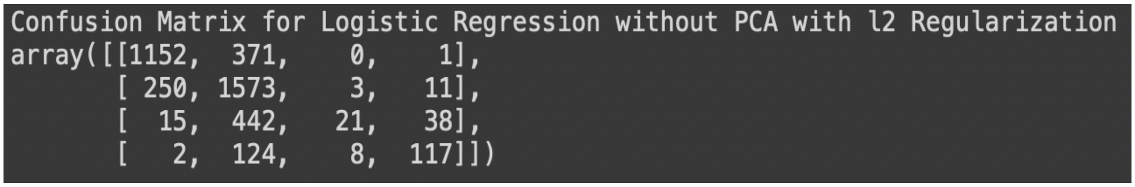
Logistic Regression l2 without PCA results:

Evaluation:



```
Accuracy on labeled data: 0.6936
Precision: 0.7032
Recall: 0.6936
F1-score: 0.6547
```

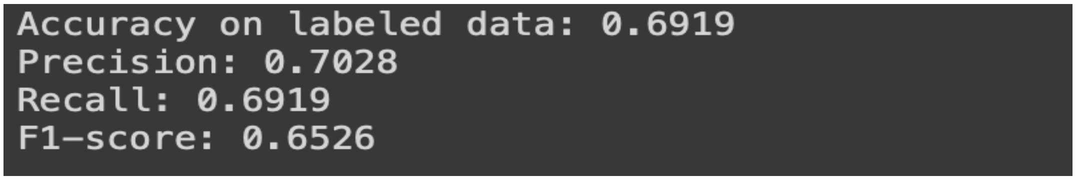
Confusion Matrics:



```
Confusion Matrix for Logistic Regression without PCA with l2 Regularization
array([[1152,  371,    0,    1],
       [ 250, 1573,    3,   11],
       [  15,  442,   21,   38],
       [   2,  124,    8,  117]])
```

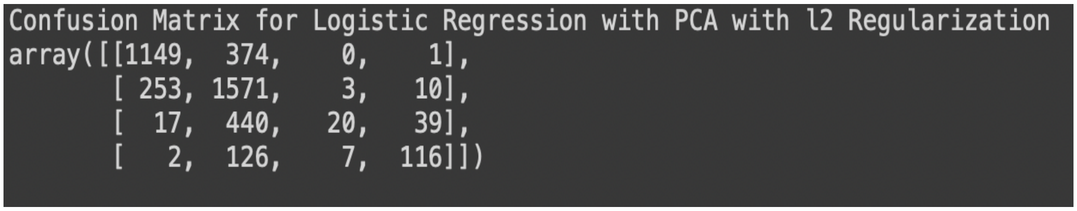
Logistic Regression l2 with PCA results:

Evaluation:



```
Accuracy on labeled data: 0.6919
Precision: 0.7028
Recall: 0.6919
F1-score: 0.6526
```

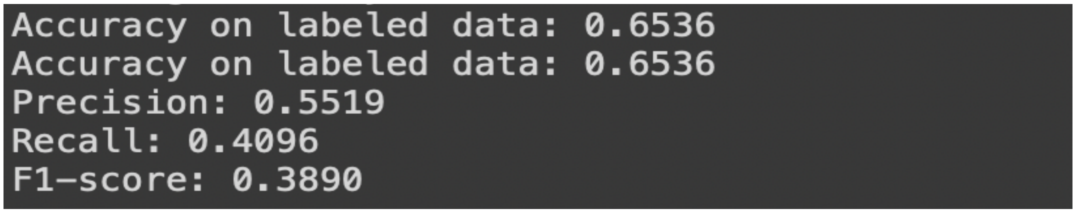
Confusion Metrics:



```
Confusion Matrix for Logistic Regression with PCA with l2 Regularization
array([[1149,  374,    0,    1],
       [ 253, 1571,    3,   10],
       [  17,  440,   20,   39],
       [   2,  126,    7,  116]])
```

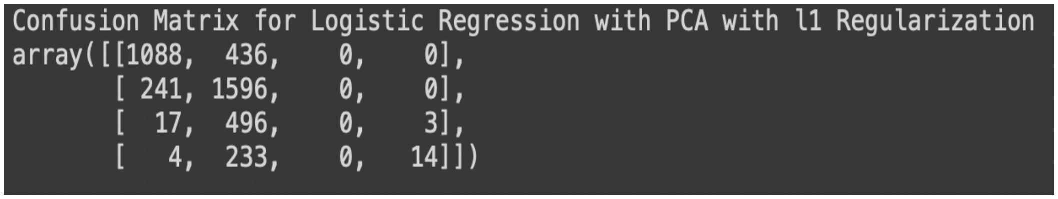
Logistic Regression l1 with PCA results:

Evaluation:



```
Accuracy on labeled data: 0.6536
Accuracy on labeled data: 0.6536
Precision: 0.5519
Recall: 0.4096
F1-score: 0.3890
```

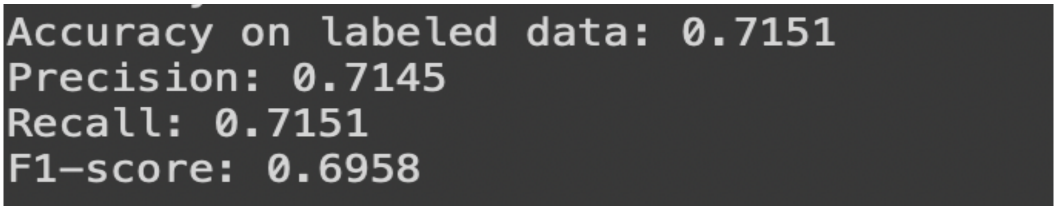
Confusion Metrics:



```
Confusion Matrix for Logistic Regression with PCA with l1 Regularization
array([[1088,  436,    0,    0],
       [ 241, 1596,    0,    0],
       [  17,  496,    0,    3],
       [   4,  233,    0,   14]])
```

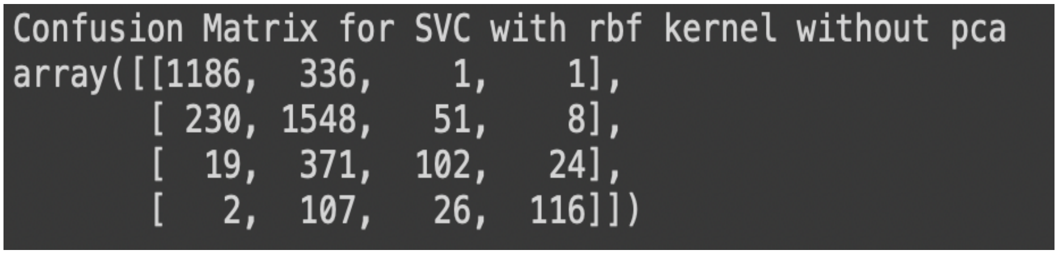
SVM rbf kernel without PCA results:

Evaluation:



```
Accuracy on labeled data: 0.7151
Precision: 0.7145
Recall: 0.7151
F1-score: 0.6958
```

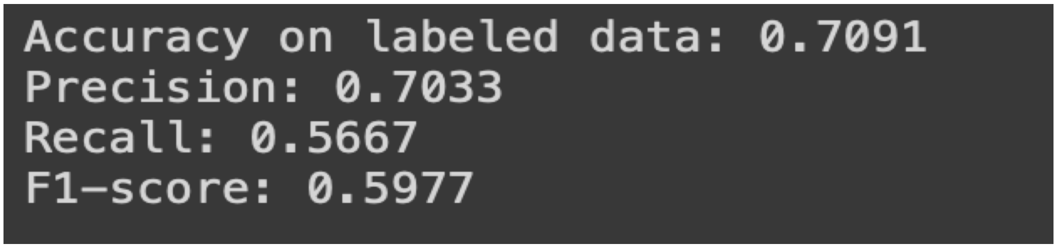
Confusion Metrics:



```
Confusion Matrix for SVC with rbf kernel without pca
array([[1186,  336,    1,    1],
       [ 230, 1548,   51,    8],
       [  19,  371,  102,   24],
       [   2,  107,   26,  116]])
```

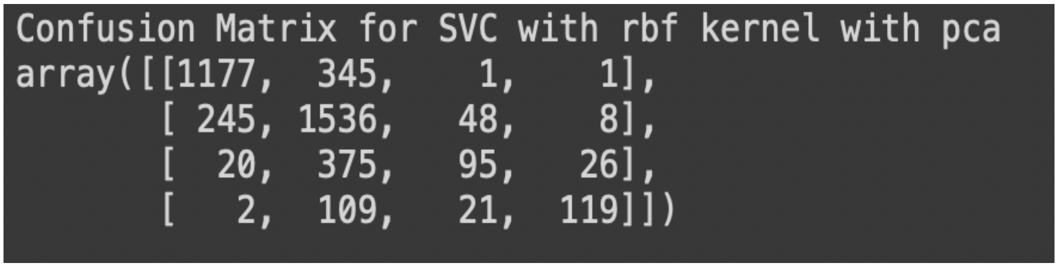
SVM rbf kernel with PCA results:

Evaluation:



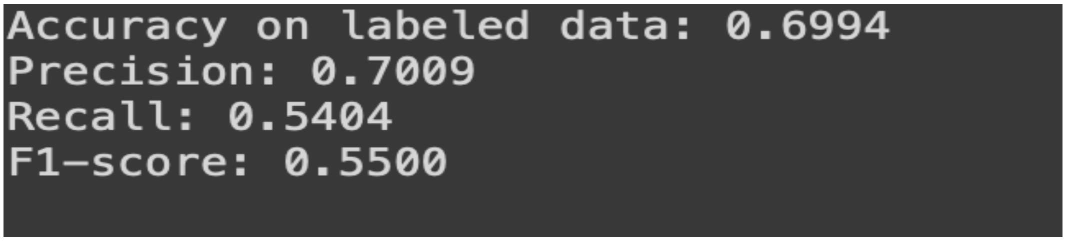
```
Accuracy on labeled data: 0.7091
Precision: 0.7033
Recall: 0.5667
F1-score: 0.5977
```

Confusion Metrics:



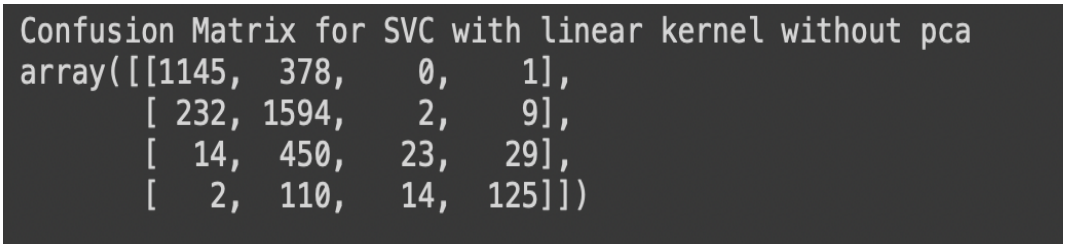
```
Confusion Matrix for SVC with rbf kernel with pca
array([[1177,  345,    1,    1],
       [ 245, 1536,   48,    8],
       [  20,  375,   95,   26],
       [   2,  109,   21,  119]])
```

SVM linear without PCA results:
Evaluation:



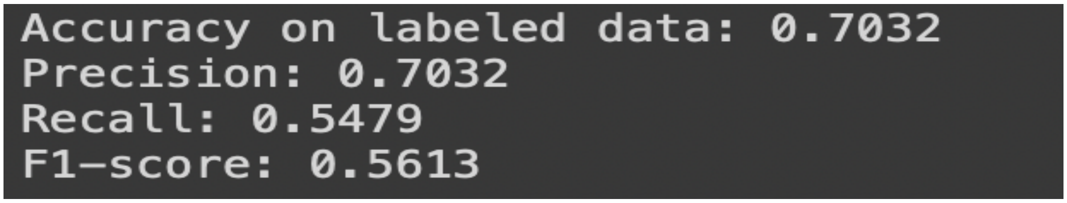
```
Accuracy on labeled data: 0.6994
Precision: 0.7009
Recall: 0.5404
F1-score: 0.5500
```

Confusion Metrics:



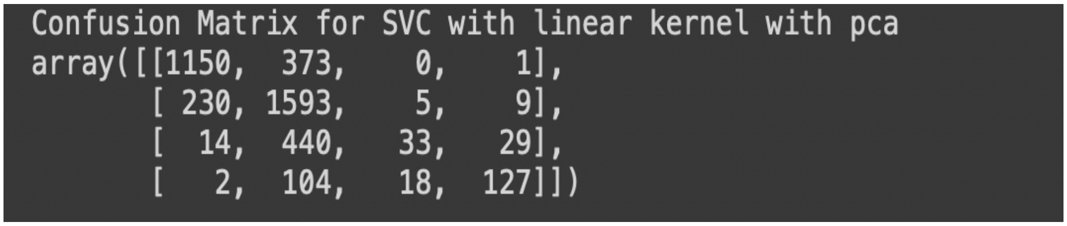
```
Confusion Matrix for SVC with linear kernel without pca
array([[1145,  378,    0,    1],
       [ 232, 1594,    2,    9],
       [  14,  450,   23,   29],
       [   2,  110,   14,  125]])
```

SVM linear kernel with PCA results:
Evaluation:



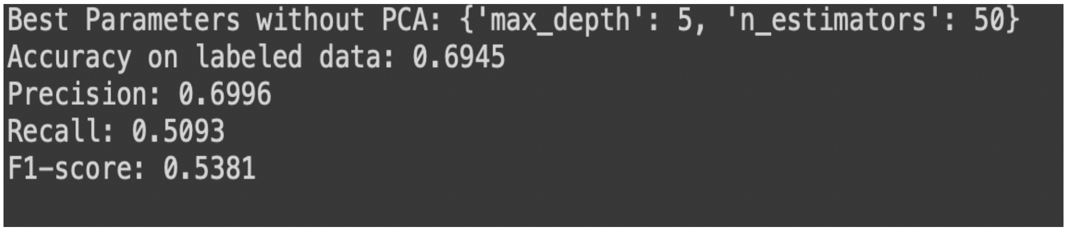
```
Accuracy on labeled data: 0.7032
Precision: 0.7032
Recall: 0.5479
F1-score: 0.5613
```

Confusion Metrics:



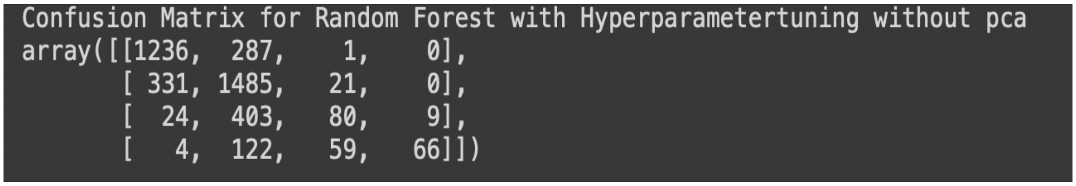
```
Confusion Matrix for SVC with linear kernel with pca
array([[1150,  373,    0,    1],
       [ 230, 1593,    5,    9],
       [  14,  440,   33,   29],
       [   2,  104,   18,  127]])
```

Random Forest without PCA result:
Evaluation:



```
Best Parameters without PCA: {'max_depth': 5, 'n_estimators': 50}
Accuracy on labeled data: 0.6945
Precision: 0.6996
Recall: 0.5093
F1-score: 0.5381
```

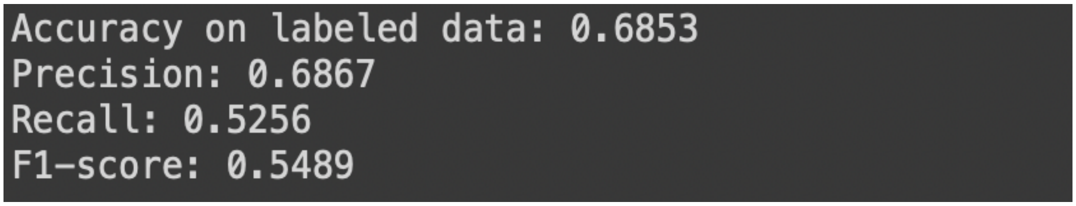
Confusion metrics:



```
Confusion Matrix for Random Forest with Hyperparametertuning without pca
array([[1236,  287,    1,    0],
       [ 331, 1485,   21,    0],
       [  24,  403,   80,    9],
       [   4,  122,   59,   66]])
```

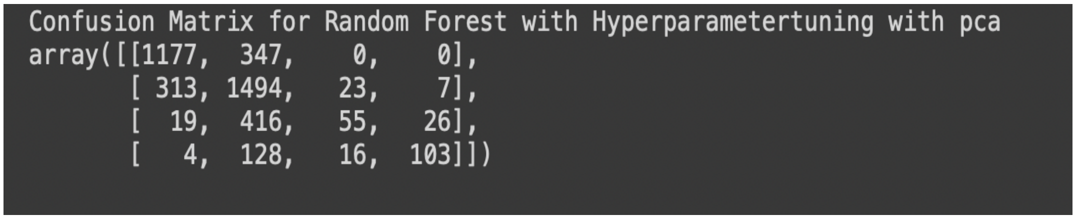
Random Forest with PCA result:

Evaluation:



```
Accuracy on labeled data: 0.6853
Precision: 0.6867
Recall: 0.5256
F1-score: 0.5489
```

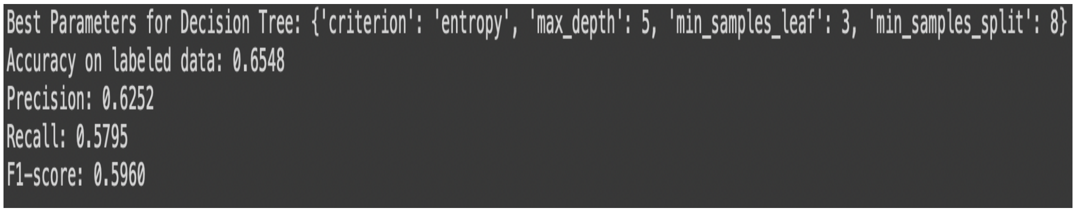
Confusion Metrics



```
Confusion Matrix for Random Forest with Hyperparametertuning with pca
array([[1177,  347,    0,    0],
       [ 313, 1494,   23,    7],
       [  19,  416,   55,   26],
       [   4,  128,   16,  103]])
```

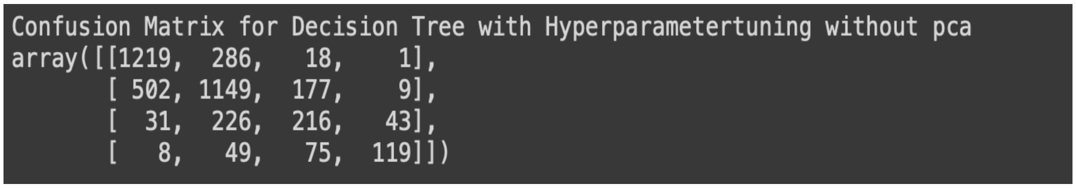
Decision Tree without PCA result:

Evaluation:



```
Best Parameters for Decision Tree: {'criterion': 'entropy', 'max_depth': 5, 'min_samples_leaf': 3, 'min_samples_split': 8}
Accuracy on labeled data: 0.6548
Precision: 0.6252
Recall: 0.5795
F1-score: 0.5960
```

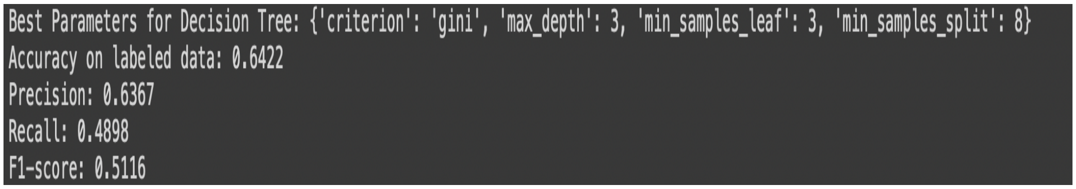
Confusion Metrics:



```
Confusion Matrix for Decision Tree with Hyperparametertuning without pca
array([[1219,  286,   18,    1],
       [ 502, 1149,  177,    9],
       [  31,  226,  216,   43],
       [   8,   49,   75,  119]])
```

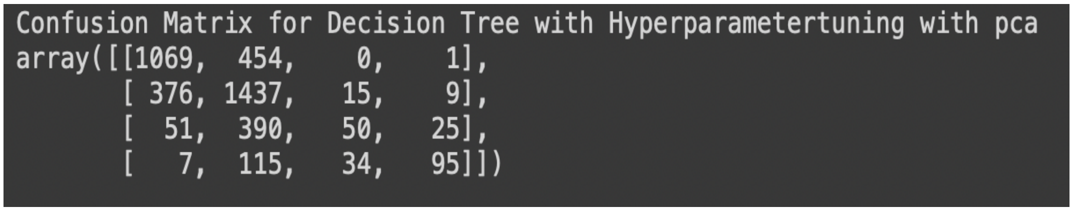
Decision Tree with PCA result:

Evaluation:



```
Best Parameters for Decision Tree: {'criterion': 'gini', 'max_depth': 3, 'min_samples_leaf': 3, 'min_samples_split': 8}
Accuracy on labeled data: 0.6422
Precision: 0.6367
Recall: 0.4898
F1-score: 0.5116
```

Confusion Metrics:



```
Confusion Matrix for Decision Tree with Hyperparametertuning with pca
array([[1069,  454,    0,    1],
       [ 376, 1437,   15,    9],
       [  51,  390,   50,   25],
       [   7,  115,   34,   95]])
```

Let's summarize the key evaluation metrics for each model:

- Logistic Regression l1 without PCA: Moderate performance across metrics.
- Logistic Regression l2 without PCA and Logistic Regression l2 with PCA: Slightly improved performance compared to M1.
- Logistic Regression l1 with PCA: Lowest performance among the models evaluated.
- SVM rbf kernel without PCA and SVM rbf kernel with PCA: Moderate to good performance in most metrics.
- SVM linear without PCA to Decision Tree with PCA: Variable performance
- The confusion matrix provides a breakdown of predictions versus actual labels for each class. It helps in understanding the model's behavior concerning misclassifications.
- A good-performing model typically has higher values along the diagonal (top-left to bottom-right), indicating a higher number of true positives. Lower off-diagonal values represent fewer false predictions.

V. Conclusion:

- Based on the provided evaluation metrics and confusion matrices SVM rbf kernel without PCA appears to perform relatively better across multiple metrics, demonstrating higher accuracy, precision, recall, and F1-score compared to other models.
- Decision Tree without PCA and Decision Tree with PCA exhibit more widespread misclassifications across various classes, contributing to lower performance.
- Therefore, **SVM rbf kernel without PCA** seems to be the **best-performing model** among those listed based on the provided evaluation metrics and confusion matrices. It shows a more balanced and accurate classification across different classes compared to the others.
- The relatively lower accuracies are anticipated as the problem in hand is semi-supervised problem.
- PCA didn’t help much in improving the accuracy for the models.