**DWDM Submission Template**

|  |  |
| --- | --- |
| All Roll Numbers of your Team: | 18MIS7250 18BCD7118 |
| Student names in your team : | Amit Kumar Sahu Ghanta Syam Venkat |
| Contribution by each team member in percentage to complete this project: | 90 10 |
| Slot (L1/L2/L4): | L4 |
| **Title of the Project:** | Census Income Analysis |
| **Objective of the Project (What exactly the project is about?)** | To predict if an individual makes greater or less than $50000 per year |

|  |  |  |
| --- | --- | --- |
| **Dataset Link** | **Number of rows and Columns** | **About columns** |
| [**https://www.mldata.io/dataset-details/census\_income/**](https://www.mldata.io/dataset-details/census_income/) | Rows:  6842  Columns:  15 | **Number of Categorical columns:** 6  **Number of Integer/Float Columns:** 6  **Number of Pure String Columns:** 3 |
|  |  | **Unique Values in each Column:**  age 70  workclass 9  fnlwgt 6104  education 16  education\_num 16  marital\_status 7  occupation 15  relationship 6  race 5  sex 2  capital\_gain 92  capital\_loss 63  hours\_per\_week 79  native\_country 40  income\_level 2 |

|  |  |  |
| --- | --- | --- |
| **Challenges identified in the project** | **How did you address that challenge?** | **References** |
| Dump values ( ‘?’ ) | Replacing it with nan | <https://towardsdatascience.com/data-preprocessing-concepts-fa946d11c825> |
| NAN values (work class & occupation) | Dropping rows with nan values | <https://towardsdatascience.com/data-preprocessing-concepts-fa946d11c825> |
| Categorical data | Encoding categorical data | <https://towardsdatascience.com/data-preprocessing-concepts-fa946d11c825> |
| NAN values in native country | Replaced missing values with mode of the column | <https://towardsdatascience.com/data-preprocessing-concepts-fa946d11c825> |
| Highly Overfitted | Pruning & Pre-processing | <https://machinelearningmastery.com/dimensionality-reduction-algorithms-with-python/> |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **Without Pre-processing- Different Algorithms ( minimal encoding)** | **Performance(Accuracy/other confusion matrix measures)** | **Which model worked well on the test data and WHY?** |
| **Algorithm 1-( Naive Bayes )** | accuracy score: 1.0000   |  |  |  | | --- | --- | --- | |  | 0 | 1 | | precision | 1.00 | 1.00 | | recall | 1.00 | 1.00 | | f1-score | 1.00 | 1.00 | | support | 1041 | 328 |   Confusion Matrix:  [[1041 0]  [ 0 328]]  ROC AUC: 1.0000 | After applying Ordinal Encoding 🡪 Decision trees work best with this training dataset as they comparatively take less time to execute than other methods. Also we have not applied further algorithm because the highest rate of accuracy, that is 100% is achieved.  Decision trees are a nonparametric machine learning algorithm that is very flexible and is subject to overfitting training data.  To Overcome overfitting, we have performed pre-processing techniques to cut off useless information. This problem can be addressed by pruning a tree after it has learned in order to remove some of the detail it has picked up |
| **Algorithm 2-( KNN )** | accuracy score: 0.8311   |  |  |  | | --- | --- | --- | |  | 0 | 1 | | precision | 0.80 | 0.58 | | recall | 0.91 | 0.34 | | f1-score | 0.86 | 0.43 | | support | 941 | 320 |   Confusion Matrix:  [[861 80]  [211 109]]  ROC AUC: 0.6278 |
| **Algorithm 3-( SVM )** | accuracy score: 1.0000   |  |  |  | | --- | --- | --- | |  | 0 | 1 | | precision | 0.75 | 0 | | recall | 1.00 | 0 | | f1-score | 0.856 | 0 | | support | 1041 | 328 |   Confusion Matrix:  [[1041 0]  [ 328 0]]  ROC AUC: 0.5000 |
| **Algorithm 4-( Decision Trees )** | accuracy score: 1.0000   |  |  |  | | --- | --- | --- | |  | 0 | 1 | | precision | 1.00 | 1.00 | | recall | 1.00 | 1.00 | | f1-score | 1.00 | 1.00 | | support | 1041 | 328 |   Confusion Matrix:  [[1041 0]  [ 0 328]]  ROC AUC: 1.0000 |
| **Algorithm 5-( Random Forest )** | accuracy score: 1.0000   |  |  |  | | --- | --- | --- | |  | 0 | 1 | | precision | 1.00 | 1.00 | | recall | 1.00 | 1.00 | | f1-score | 1.00 | 1.00 | | support | 1041 | 328 |   Confusion Matrix:  [[1041 0]  [ 0 328]]  ROC AUC: 1.0000 |
| **Algorithm 6-( Bagging )** | accuracy score: 1.0000   |  |  |  | | --- | --- | --- | |  | 0 | 1 | | precision | 1.00 | 1.00 | | recall | 1.00 | 1.00 | | f1-score | 1.00 | 1.00 | | support | 1041 | 328 |   Confusion Matrix:  [[1041 0]  [ 0 328]]  ROC AUC: 1.0000 |
| **Algorithm 7-( Gradient Boosting Algorithm)** | accuracy score: 1.0000   |  |  |  | | --- | --- | --- | |  | 0 | 1 | | precision | 1.00 | 1.00 | | recall | 1.00 | 1.00 | | f1-score | 1.00 | 1.00 | | support | 1041 | 328 |   Confusion Matrix:  [[1041 0]  [ 0 328]]  ROC AUC: 1.0000 |  |
| **Algorithm 8-( XG Boost )** | accuracy score: 1.0000   |  |  |  | | --- | --- | --- | |  | 0 | 1 | | precision | 1.00 | 1.00 | | recall | 1.00 | 1.00 | | f1-score | 1.00 | 1.00 | | support | 1041 | 328 |   Confusion Matrix:  [[1041 0]  [ 0 328]]  ROC AUC: 1.0000 |  |

|  |  |  |
| --- | --- | --- |
| **Which Pre-processing technique you applied?** | **Why you applied that pre-processing Technique?** | **References** |
| Checking for missing values and removing the rows with missing values | Missing values may lead to weak training of the model also some algorithms don’t work with missing values. | <https://towardsdatascience.com/data-pre-processing-techniques-you-should-know-8954662716d6> |
| Checking for categorical data and converting them to integer with ordinal encoding. | It’s difficult to implement algorithms with categorical data so ordinal encoding to preserve order of categorical data and convert it to int. | <https://towardsdatascience.com/data-pre-processing-techniques-you-should-know-8954662716d6> |
| Checking for categorical data and converting them to integer with label encoding. | label encoding for categorical data, where there's no order in data | <https://towardsdatascience.com/data-pre-processing-techniques-you-should-know-8954662716d6> |
| Replacing NaN values of a column with mode of the column | Since the feature we are considering is important for our model in order to increase efficiency and reduce data loss we replace with the mode of the column. | <https://towardsdatascience.com/data-pre-processing-techniques-you-should-know-8954662716d6> |
| Standardizing the data using min max scalar | Transform features by scaling each feature to a given range. This estimator scales and translates each feature individually such that it is in the given range on the training set | <https://towardsdatascience.com/data-pre-processing-techniques-you-should-know-8954662716d6> |
| Standardizing the data using standard scalar | It will transform your data such that its distribution will have a mean value 0 and standard deviation of 1 | <https://towardsdatascience.com/data-pre-processing-techniques-you-should-know-8954662716d6> |
| Data splitting 80-20 | 80% of effects come from 20% of causes keeping this principle in mind so that model is efficient. | <https://towardsdatascience.com/data-pre-processing-techniques-you-should-know-8954662716d6> |

|  |  |  |
| --- | --- | --- |
| **Algorithms Applied after preprocessing [Label Encoding)** | **Performance(Accuracy/other confusion matrix measures)** | **Which model worked well on the test data and WHY?** |
| **Algorithm 1-( Naive Bayes with min max scalar )** | accuracy score: 1.0000   |  |  |  | | --- | --- | --- | |  | 0 | 1 | | precision | 0.79 | 0.71 | | recall | 0.95 | 0.32 | | f1-score | 0.87 | 0.44 | | support | 943 | 343 |   Confusion Matrix:  [[898 45]  [234 109]]  ROC AUC: 0.6350 | After applying label encoding -> standard scalar -> stacking -> meta classifier -> random forest and decision trees gave the highest performance of around 88.1% ROC accuracy.  This is due to base learners also the combination of two algorithms. Which resulted in improved accuracy and overcoming overfitting. |
| **Algorithm 2-( KNN with min max scalar )** | accuracy score: 1.0000   |  |  |  | | --- | --- | --- | |  | 0 | 1 | | precision | 0.85 | 0.66 | | recall | 0.90 | 0.56 | | f1-score | 0.87 | 0.61 | | support | 943 | 343 |   Confusion Matrix:  [[844 99]  [151 192]]  ROC AUC: 0.7274 |
| **Algorithm 3-( KNN with min max scalar and best k value i.e 14)** | accuracy score: 1.0000   |  |  |  | | --- | --- | --- | |  | 0 | 1 | | precision | 0.84 | 0.72 | | recall | 0.92 | 0.52 | | f1-score | 0.88 | 0.61 | | support | 943 | 343 |   Confusion Matrix:  [[872 71]  [163 180]]  ROC AUC: 0.7247 |
| **Algorithm-(**Naive Bayes using min max scalar  **)** | accuracy score: 0.8058   |  |  |  | | --- | --- | --- | |  | 0 | 1 | | precision | 0.82 | 0.71 | | recall | 0.95 | 0.36 | | f1-score | 0.88 | 0.48 | | support | 3872 | 1268 |   Confusion Matrix:  [[3687 185]  [ 813 455]]  ROC AUC: 0.6555 |  |
| **Algorithm-(**KNN with min max scaling  **)** | accuracy score: 0.8056   |  |  |  | | --- | --- | --- | |  | 0 | 1 | | precision | 0.85 | 0.66 | | recall | 0.90 | 0.56 | | f1-score | 0.87 | 0.61 | | support | 943 | 343 |   Confusion Matrix:  [[844 99]  [151 192]]  ROC AUC: 0.7274 |  |
| **Algorithm-(KNN with min max scalar and best k value i.e 14)** | accuracy score: 0.8180   |  |  |  | | --- | --- | --- | |  | 0 | 1 | | precision | 0.84 | 0.72 | | recall | 0.92 | 0.52 | | f1-score | 0.88 | 0.61 | | support | 943 | 343 |   Confusion Matrix:  [[872 71]  [163 180]]  ROC AUC: 0.7247 |  |
| **Algorithm-(**SVM using min max scalar  **)** | accuracy score: 0.8180   |  |  |  | | --- | --- | --- | |  | 0 | 1 | | precision | 0.84 | 0.72 | | recall | 0.92 | 0.52 | | f1-score | 0.88 | 0.61 | | support | 943 | 343 |   Confusion Matrix:  [[872 71]  [163 180]]  ROC AUC: 0.7247 |  |
| **Algorithm-(**Decision trees using min max classfier  **)** | accuracy score: 0.7799   |  |  |  | | --- | --- | --- | |  | 0 | 1 | | precision | 0.84 | 0.60 | | recall | 0.87 | 0.55 | | f1-score | 0.85 | 0.57 | | support | 943 | 343 |   Confusion Matrix:  [[816 127]  [156 187]]  ROC AUC: 0.7053 |  |
| **ALgorithm-(**Random Forest with 100 trees**)** | accuracy score: 0.8289   |  |  |  | | --- | --- | --- | |  | 0 | 1 | | precision | 0.86 | 0.71 | | recall | 0.91 | 0.60 | | f1-score | 0.89 | 0.85 | | support | 943 | 343 |   Confusion Matrix:  [[859 84]  [136 207]]  ROC AUC: 0.7572 |  |
| **Algorithm-(**Gradient Boost min max scalar**)** | accuracy score: 0.8468   |  |  |  | | --- | --- | --- | |  | 0 | 1 | | precision | 0.88 | 0.75 | | recall | 0.92 | 0.64 | | f1-score | 0.90 | 0.69 | | support | 943 | 343 |   Confusion Matrix:  [[868 75]  [122 221]]  ROC AUC: 0.7824 |  |
| **Algorithm-(**Adaboost**)** | accuracy score: 0.8499   |  |  |  | | --- | --- | --- | |  | 0 | 1 | | precision | 0.87 | 0.78 | | recall | 0.94 | 0.60 | | f1-score | 0.90 | 0.68 | | support | 943 | 343 |   Confusion Matrix:  [[886 57]  [136 207]]  ROC AUC: 0.7715 |  |
| **Algorithm-(**Adaboost - Random Forest min max scalar**)** | accuracy score: 0.8142   |  |  |  | | --- | --- | --- | |  | 0 | 1 | | precision | 0.85 | 0.68 | | recall | 0.91 | 0.56 | | f1-score | 0.88 | 0.62 | | support | 943 | 343 |   Confusion Matrix:  [[854 89]  [150 193]]  ROC AUC: 0.7342 |  |
|  |  |  |
| **ALgorithm-(**XGBoost classifier**)** | accuracy score: 0.8546   |  |  |  | | --- | --- | --- | |  | 0 | 1 | | precision | 0.88 | 0.78 | | recall | 0.93 | 0.64 | | f1-score | 0.90 | 0.64 | | support | 943 | 343 |   Confusion Matrix:  [[881 62]  [125 218]]  ROC AUC: 0.7849 |  |
| **Algorithm-(**Naive Bayes with standard scaling**)** | accuracy score: 0.7830   |  |  |  | | --- | --- | --- | |  | 0 | 1 | | precision | 0.79 | 0.71 | | recall | 0.95 | 0.32 | | f1-score | 0.87 | 0.44 | | support | 943 | 343 |   Confusion Matrix:  [[898 45]  [234 109]]  ROC AUC: 0.6350 |  |
| **Algorithm-(**KNN with standard scaling**)** | accuracy score: 0.8180   |  |  |  | | --- | --- | --- | |  | 0 | 1 | | precision | 0.84 | 0.73 | | recall | 0.93 | 0.50 | | f1-score | 0.88 | 0.60 | | support | 943 | 343 |   Confusion Matrix:  [[880 63]  [171 172]]  ROC AUC: 0.7173 |  |
| **Algorithm-(**SVM with standard scaling**)** | accuracy score: 0.8305   |  |  |  | | --- | --- | --- | |  | 0 | 1 | | precision | 0.84 | 0.77 | | recall | 0.94 | 0.52 | | f1-score | 0.89 | 0.62 | | support | 943 | 343 |   Confusion Matrix:  [[888 55]  [163 180]]  ROC AUC: 0.7332 |  |
| **Algorithm-(Gini index and entropy)** | accuracy score:  82.42612752721618   |  |  |  | | --- | --- | --- | |  | 0 | 1 | | precision | 0.84 | 0.76 | | recall | 0.94 | 0.49 | | f1-score | 0.89 | 0.60 | | support | 943 | 343 |   Confusion Matrix:  [[891 52]  [174 169]]  accuracy score: 83.2814930015552   |  |  |  | | --- | --- | --- | |  | 0 | 1 | | precision | 0.84 | 0.76 | | recall | 0.94 | 0.54 | | f1-score | 0.89 | 0.63 | | support | 943 | 343 |   Confusion Matrix:  [[885 58]  [157 186]] |  |
| **Algorithm-(**gini index and entropy both same values**)** | accuracy score: 0.8095   |  |  |  | | --- | --- | --- | |  | 0 | 1 | | precision | 0.81 | 0.81 | | recall | 0.97 | 0.37 | | f1-score | 0.88 | 0.51 | | support | 943 | 343 |   Confusion Matrix:  [[913 30]  [215 128]]  ROC AUC: 0.6707 |  |
| **Algorithm-(**random forest (ensemble) with standard scaling**)** | accuracy score: 0.8281   |  |  |  | | --- | --- | --- | |  | 0 | 1 | | precision | 0.86 | 0.71 | | recall | 0.91 | 0.85 | | f1-score | 0.89 | 0.65 | | support | 943 | 343 |   Confusion Matrix:  [[861 82]  [139 204]]  ROC AUC: 0.7539 |  |
| **Algorithm-(Bagging with Standard Scalling)** | accuracy score: 0.8025   |  |  |  | | --- | --- | --- | |  | 0 | 1 | | precision | 0.82 | 0.73 | | recall | 0.94 | 0.41 | | f1-score | 0.88 | 0.53 | | support | 943 | 343 |   Confusion Matrix:  [[890 53]  [201 142]]  ROC AUC: 0.6789 |  |
| **Algorithm-(Gradient boosting with random scaling)** | accuracy score: 0.8538   |  |  |  | | --- | --- | --- | |  | 0 | 1 | | precision | 0.86 | 0.83 | | recall | 0.96 | 0.57 | | f1-score | 0.91 | 0.67 | | support | 943 | 343 |   Confusion Matrix:  [[904 39]  [149 194]]  ROC AUC: 0.7621 |  |
| **Algorithm-(XG boost with standard scaling)** | accuracy score: 0.8468   |  |  |  | | --- | --- | --- | |  | 0 | 1 | | precision | 0.88 | 0.75 | | recall | 0.92 | 0.64 | | f1-score | 0.90 | 0.69 | | support | 943 | 343 |   Confusion Matrix:  [[868 75]  [122 221]]  ROC AUC: 0.7824 |  |
| **Algorithm-(**adaboost -> random forest with standard scaling**)** | accuracy score: 0.8142   |  |  |  | | --- | --- | --- | |  | 0 | 1 | | precision | 0.85 | 0.69 | | recall | 0.91 | 0.56 | | f1-score | 0.88 | 0.62 | | support | 943 | 343 |   Confusion Matrix:  [[855 88]  [151 192]]  ROC AUC: 0.7332 |  |
| **Algorithm-(** Meta Classifier- as Logistic Regression and ensembles as "Decision Tree", "SVM" and "Random Forest**)** | Confusion Matrix:  [[876 65]  [115 205]]  ROC AUC: 0.7858 |  |
| **Algorithm-(** Stacking -> Decision trees, Random forest, SVM & Xg boost**)** | Confusion Matrix:  [[882 59]  [119 201]]  ROC AUC: 0.7827 |  |
| **Algorithm-(** PCA with 7 components**)** |  | Accuracy: 0.820 (0.033) |
| **Algorithm-(** PCA with 2 components**)** |  | Accuracy: 0.794 (0.040) |
|  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Pre-processing technique name?** | **Data Mining Algorithm you applied?** | **Performance(Accuracy/other confusion matrix measures)**  **(Before pre-processing) (After Pre-processing)** | | **Which model worked well on the test data and WHY?** |
| **Categorical Encoding Mechanism – Min Max Scalar** | **Naïve Bayes** | Confusion Matrix:  [[941 0]  [ 0 320]]  ROC AUC: 1.0000 | ROC AUC: 0.6350 | As the dataset was highly overfitted after pre-processing the model seems to have better and near to accurate performance.  Stacking Meta Classifier with random classifier and decision trees gave the highest accuracy among all because it’s a combination of all 2 algorithms as base learners and giving better performance. |
| **Categorical Encoding Mechanism - Min Max Scalar** | **KNN** | Confusion Matrix:  [[861 80]  [211 109]]  ROC AUC: 0.6278 | ROC AUC: 0.7173  KNN with min max scalar and best k value i.e 14  ROC AUC: 0.7247 |
| **Categorical Encoding Mechanisms- Min Max Scalar** | **SVM** | Confusion Matrix:  [[941 0]  [320 0]]  ROC AUC: 0.5000 | ROC AUC: 0.7332 |
| **Categorical Encoding Mechanisms-** min max classfier | Decision trees | Confusion Matrix:  [[816 127]  [156 187]]  ROC AUC: 0.7053 | ROC AUC: 0.7823 |
| **Categorical Encoding Mechanisms** | Random Forest with 100 trees | Confusion Matrix:  [[859 84]  [136 207]]  ROC AUC: 0.7572 | ROC AUC: 0.7539 |
| **Categorical Encoding Mechanisms-**min max scalar | Gradient Boost | Confusion Matrix:  [[868 75]  [122 221]]  ROC AUC: 0.7824 | ROC AUC: 0.7636 |  |
| **Categorical Encoding Mechanisms** | Adaboost | Confusion Matrix:  [[886 57]  [136 207]]  ROC AUC: 0.7342 | ROC AUC: 0.7715 |  |
| **Categorical Encoding Mechanisms** | XGBoost classifier | Confusion Matrix:  [[877 66]  [132 211]]  ROC AUC: 0.7726 | ROC AUC: 0.7824 |  |
| **Categorical Encoding Mechanisms-** | Gini index and entropy | Gini Index  ROC AUC: 0.6707  Entropy  ROC AUC: 0.6521 | Gini Index  ROC AUC: 0.8328  Entropy  ROC AUC: 0.8267 |  |
| **Categorical Encoding Mechanisms-Standard scaling** | random forest (ensemble) | Confusion Matrix:  [[861 82]  [139 204]]  ROC AUC: 0.7332 | ROC AUC: 0.7539 |  |
| **Categorical Encoding Mechanisms-Standard scaling** | Bagging |  | ROC AUC: 0.6789 |  |
| **Categorical Encoding Mechanisms-Standard scaling** | Stacking -> meta classifier -> Random forest & decision trees | ROC AUC: 0.8800 | ROC AUC: 0.8817 |  |
| **Dimension Reduction** | PCA with 7 components |  | ROC AUC: 0.8234 |  |
| **Single Value Decomposition** | SVD |  | ROC AUC: 0.8421 |  |

|  |
| --- |
| **Summary:** |
| **Number of Pre-processing Techniques applied with their names:**   1. **Dealing with missing values** 2. **Ordinal Encoding** 3. **Cleaning Data – mode filling** 4. **Label Encoding** 5. **Min Max Scalar** 6. **Standard Scalar** 7. **Dimension Reduction**   **Number of Data Mining Algorithms applied with their names:**   1. **Naïve Byes** 2. **KNN** 3. **Decision Trees** 4. **Decision Trees -> gini index** 5. **Decision Trees -> entropy** 6. **Decision Trees -> Caliberated Classifier -> sigmoid function** 7. **Random Forest** 8. **SVM -> rbf** 9. **Bagging** 10. **Gradient Boosting** 11. **Correlation Analysis** 12. **XG Boost** 13. **Adaboost** 14. **Adaboost -> Random Forest** 15. **Logistic Regression** 16. **Stacking -> Meta Classifier -> Decision trees, Random forest** 17. **Stacking -> Meta Classifier -> Decision trees, Random forest & Xg boost** 18. **Stacking -> Single Classifier** 19. **Logistic Regression SGDClassifier** 20. **Principal Component Analysis** 21. **Singular Value Decomposition** |
| **Which algorithm showed highest performance after all pre-processing techniques and WHY?:**  Label Encoding -> Standard Scaling -> Ensemble Learning -> Stacking -> Meta Classifier- as Logistic Regression and ensembles as "Decision Tree" and "Random Forest" with ROC accuracy of 88.7%.  As combining different predictive models improves accuracy. stacking is appropriate when multiple different machine learning models have skill on a dataset, but have skill in different ways.  Followed by  Label Encoding -> Min Max Scaling -> Ensemble Learning -> Stacking -> Meta Classifier- as Logistic Regression and ensembles as "Decision Tree" and "Random Forest" with ROC accuracy of 88.5%. |
| **Conclusion-Write in your own words:**  Overall this project was challenging and I got an opportunity to learn and brush up my data mining and machine leaning algorithms, also learned when to apply which algorithm and consider the efficiency of algorithm not only on the basis of accuracy but also other measures like f1-score, recall, ROC accuracy etc  It was fun exploring the trends in the model when changing hyperparameters and playing around several algorithms. |