Income Prediction using Pyspark and MLlib

Big Data DSM010
March 2022

1. Topic proposal

Introduction:

The aim of this project is to predict an individual's income accurately with a set of input features. The method is to use supervised leaning algorithms. The income feature is used to predict an individual's income level in 2 classes:

Category 0: <= $50 000 per year.

Category 1: > $50 000

Because there are only two classes in the dataset, the Machine Learning task is a binary classification one. The aim is to predict if a person earns more than $ 50k per year or not.

Technologies required to run project:

Hadoop HDFS, PySpark API, Apache Spark MLlib, Apache Spark SQL, Python.

The Adult Income Dataset:

The Adult Census Income Dataset is one created from the 1994 United States Census Bureau data. The data was sourced from the following webpage: Datasets/adult-all.csv at master · jbrownlee/Datasets (github.com)

The dataset consists of 32 561 rows and 15 features of which 7 are continuous and 8 are categorical. This dataset was uploaded to the Hadoop Distributed File System (HDFS).

Continuous features:
x, age, fnlwgt, educational-num, capital-gain, capital-loss, hours-per-week.
Age ranges from 17 – 90; educational-num gives the number of years education was taken; fnlwgt is the weight assigned to the combination of features.

Categorical features:
workclass, education, marital-status, occupation, relationship, race, gender, native-country.
Education is the highest level of education obtained. Workclass is the class of work to which an individual belongs.

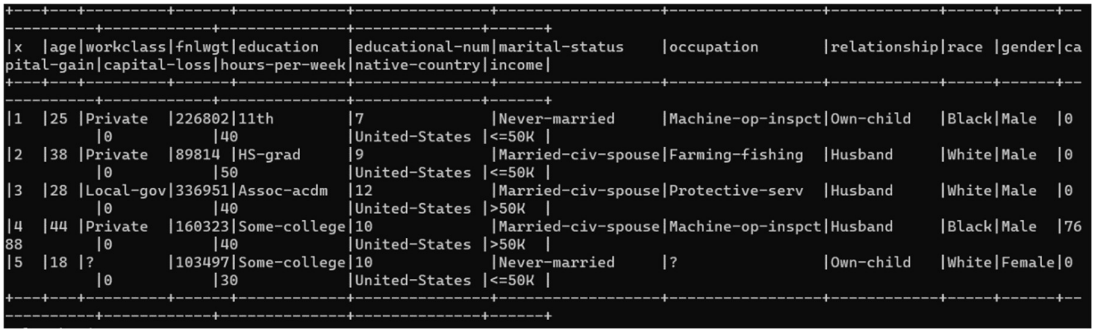
Exploratory Data Analysis (EDA):

Exploring the dataset revealed that:

- Null values are to be found in workclass, capital-gain, marital-status, occupation, capital-loss, hours-per-week, race and gender.
- Data is skewed to income category 0 at a ratio of 75/25%.
- There are 16 columns and 48842 rows in the adult dataset.

- The “x” column is an ID column. The column “fnlwgt” is a weighted column, neither of these will be used in the analysis.

- There is missing data marked with “?”. This can be seen in Fig2. below:



x	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	capital-loss	hours-per-week	native-country	income
1	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child	Black	Male	0	0	40	United-States	<=50K
2	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	0	0	50	United-States	<=50K
3	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male	0	0	40	United-States	>50K
4	44	Private	160323	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688	0	40	United-States	>50K
5	18	?	103497	Some-college	10	Never-married	?	Own-child	White	Female	0	0	30	United-States	<=50K

Fig 1. Print of the first 5 rows of dataset

Observations from continuous variables:

- The greater the age and education level and the number of hours worked, the higher the income level.

- There is no variation in the capital-gain and capital-loss data, so these variables won’t be a good predictor of income level unless the mean is used.

- Age and education levels both have good variance, so would be good predictors of income.

- Capital-gain and capital-loss don’t show much variance for all income levels - their means show a difference for different income levels so will also be good for prediction.

```
>>> numeric_data.corr()
                     age  capital-gain  educational-num  capital-loss  hours-per-week
age             1.000000      0.077229         0.030940      0.056944        0.071558
capital-gain    0.077229      1.000000         0.125146     -0.031441        0.082157
educational-num 0.030940      0.125146         1.000000      0.080972        0.143689
capital-loss    0.056944     -0.031441         0.080972      1.000000        0.054467
hours-per-week  0.071558      0.082157         0.143689      0.054467        1.000000
```

Fig. 2 Correlation Matrix

```
>>> numeric_data.describe()
                age  capital-gain  educational-num  capital-loss  hours-per-week
count  48842.000000  48842.000000     48842.000000  48842.000000    48842.000000
mean      38.643585   1079.067627        10.078089     87.502312       40.422382
std       13.710510   7452.018555         2.570973    403.004547       12.391444
min       17.000000      0.000000         1.000000      0.000000        1.000000
25%       28.000000      0.000000         9.000000      0.000000       40.000000
50%       37.000000      0.000000        10.000000      0.000000       40.000000
75%       48.000000      0.000000        12.000000      0.000000       45.000000
max       90.000000  99999.000000        16.000000   4356.000000       99.000000
```

Fig. 3 Descriptive Statistics – numeric data

gender	avg(capital-gain)
Female	580.7262845849802
Male	1326.2083001531394

Fig. 4 Race vs capital-gain

income	avg(hours-per-week)
<=50K	38.840048445700045
>50K	45.45289638059382

Fig. 5 Income vs hours-per-week

occupation	avg(capital-gain)
Sales	1266.8982558139535
Exec-managerial	2277.7555044364112
Prof-specialty	2745.9180168502917
Handlers-cleaners	283.41264478764048
Farming-fishing	715.9375838926175
Craft-repair	718.9520615183246
Transport-moving	441.543949044586
Priv-house-serv	185.16528925619835
Protective-serv	717.0661241098677
Other-service	217.1275644931952
Tech-support	670.5511756569848
Machine-op-inspct	315.22766379880875
Armed-Forces	486.53333333333336
?	501.15557137771407

Fig. 6 Occupation vs capital-gain (avg)

race	avg(capital-gain)
Other	983.2241379310345
Amer-Indian-Eskimo	538.9765957446808
White	1124.4797662947176
Asian-Pac-Islander	1537.2666227781435
Black	588.1923159018143

Fig. 7 Race vs capital-gain (avg)

race	avg(educational-num)
Other	8.839901477832512
Amer-Indian-Eskimo	9.38723404255319
White	10.13026196063407
Asian-Pac-Islander	10.998683344305464
Black	9.491141942369264

Fig. 8 Race vs educational-num

Observations from categorical variables:

- There are a large number of the following three variables: race = “White”; workclass = “Private” ; native-country = “United-States”
- There is only 1 record of data for native-country = “Holand-Netherlands”
- Adults with an educational level of Prof-school and Doctorate have a higher income and are more likely to have income >50k.
- Adults with an occupation Prof-speciality and Exec-managerial are also more likely to earn more than 50k.
- Self-emp and Federal-gov workclass are also more likely to earn >50k as are Asian-Pac-Islander or White people.
- A married couple are more likely to earn >50k compared to other relationship types.
  The gender variable shows that men are more likely to earn more than woman.

workclass	count
Self-emp-not-inc	3862
Local-gov	3136
State-gov	1981
Private	33906
Without-pay	21
Federal-gov	1432
Never-worked	10
?	2799
Self-emp-inc	1695

Fig. 9 Workclass vs income

race	income	count	%
White	<=50K	31155	63.79
White	>50K	10607	21.72
Black	<=50K	4119	8.43
Asian-Pac-Islander	<=50K	1110	2.27
Black	>50K	566	1.16
Amer-Indian-Eskimo	<=50K	415	0.85
Asian-Pac-Islander	>50K	409	0.84
Other	<=50K	356	0.73
Amer-Indian-Eskimo	>50K	55	0.11
Other	>50K	50	0.1

Fig. 10 Race vs income proportion

race	count
Other	406
Amer-Indian-Eskimo	470
White	41762
Asian-Pac-Islander	1519
Black	4685

Fig. 11 Race count

gender	income	count	%
Male	<=50K	22732	46.54
Female	<=50K	14423	29.53
Male	>50K	9918	20.31
Female	>50K	1769	3.62

Fig. 12 Gender vs income proportion

native-country	count
Philippines	295
Germany	206
Cambodia	28
France	38
Greece	49
Taiwan	65
Ecuador	45
Nicaragua	49
Hong	30
Peru	46
India	151
China	122
Italy	105
Holand-Netherlands	1
Cuba	138
South	115
Iran	59
Ireland	37
Thailand	30
Laos	23

only showing top 20 rows

Fig. 13 Native-country count

native-country	count	%
United-States	43832	89.74
Mexico	951	1.95
?	857	1.75
Philippines	295	0.6
Germany	206	0.42
Puerto-Rico	184	0.38
Canada	182	0.37
El-Salvador	155	0.32
India	151	0.31
Cuba	138	0.28
England	127	0.26
China	122	0.25
South	115	0.24
Jamaica	106	0.22
Italy	105	0.21
Dominican-Republic	103	0.21
Japan	92	0.19
Guatemala	88	0.18
Poland	87	0.18
Vietnam	86	0.18

only showing top 20 rows

Fig. 14 Native-country proportion

marital-status	income	count	%
Never-married	<=50K	15384	31.5
Married-civ-spouse	<=50K	12395	25.38
Married-civ-spouse	>50K	9984	20.44
Divorced	<=50K	5962	12.21
Separated	<=50K	1431	2.93
Widowed	<=50K	1390	2.85
Never-married	>50K	733	1.5
Divorced	>50K	671	1.37
Married-spouse-ab...	<=50K	570	1.17
Widowed	>50K	128	0.26
Separated	>50K	99	0.2
Married-spouse-ab...	>50K	58	0.12
Married-AF-spouse	<=50K	23	0.05
Married-AF-spouse	>50K	14	0.03

Fig. 15 Marital-status vs income

education	income	count	%
HS-grad	<=50K	13281	27.19
Some-college	<=50K	8815	18.05
Bachelors	<=50K	4712	9.65
Bachelors	>50K	3313	6.78
HS-grad	>50K	2503	5.12
Some-college	>50K	2063	4.22
11th	<=50K	1720	3.52
Assoc-voc	<=50K	1539	3.15
Masters	>50K	1459	2.99
10th	<=50K	1302	2.67
Masters	<=50K	1198	2.45
Assoc-acdm	<=50K	1188	2.43
7th-8th	<=50K	893	1.83
9th	<=50K	715	1.46
Prof-school	>50K	617	1.26
12th	<=50K	609	1.25
Assoc-voc	>50K	522	1.07
5th-6th	<=50K	482	0.99
Doctorate	>50K	431	0.88
Assoc-acdm	>50K	413	0.85

only showing top 20 rows

Fig. 16 Educational vs income proportion

Hypotheses:

The hypothesis for this project is that an individual's annual income can be accurately predicted from the following features: age, educational-num, capital-gain, capital-loss, hours-per-week, workclass, education, marital-status, occupation, relationship, race, gender, native-country.

The method of testing the hypotheses is to train different machine learning algorithms in the prediction of income using training data, then to validate the results using the test data. The Adult Census Income from the 1994 United States Census Bureau dataset was used for this project. Results from the modelling are then compared using three metrics, namely area under ROC, F1 score and accuracy to determine which algorithm is best suited for the dataset.

The overall goals of the project:

The input data is binary, meaning a Supervised learning technique is required to model the data. Specifically binary classification. Some algorithms are specifically designed for binary classification; these include Logistic Regression and Support Vector Machines.

The following models were chosen to address this binary classification problem:

- Logistic Regression
- Decision Tree
- Random Forest
- Support Vector Machines (SVMs)
- Gradient Boost

Classification predictive modelling is used to test the hypotheses. Classification in a machine learning perspective refers to a predictive modelling problem where a class label is predicted from a set of input data. In this case, the classifier must predict whether to classify the output as income > 50K or <= 50K based on a set of features.

The classification task needs a training dataset with many examples from which it can learn. The model calculates the best method of mapping input data to a particular class label. To discover which algorithm predicts the class label the most accurately involves experimenting and evaluating each model using certain metrics.

PySpark, which is an interface for Apache Spark which allows the use of Python APIs, was used to write the code needed to model the data. MLlib was used for the Machine Learning. The output is a .py file that can be run directly in a PySpark Terminal window or as a script file or on the Ubuntu Cluster with a *spark-submit* command. Hadoop Distributed File System (HDFS) is used to store and distribute the input data. MLlib is Spark's Machine Learning Library that

The models are trained on the training data using the *.fit()* function; predictions are done on the testing data using the *.predict()* functions. Models are evaluated with metrics such as accuracy, F1 and area under ROC. In some cases, Cross Validation models (CV) were also successfully run. These CV models can use up a lot of computing resources; because of this factor, only two CV models were attempted.

To achieve the classification, the following steps are essential in PySpark:

- Import the required libraries
- Create a Spark Session
- Load input data (in .csv format) from Hadoops Distributed File System
- Exploratory Data Analysis (EDA) – which includes summary statistics
- Data preprocessing – includes checking for null values and missing data; changing the data type; removing unnecessary data; checking for unbalanced data.
- Building a Spark ML Pipeline - includes String Indexer, One Hot Encode, Vector Assembler and Estimator
- Randomly Split the data (80 / 20) and set the seed to allow for replication
- Running the models and evaluating them using three metrics: Accuracy, F1 Score, Area under ROC curve (AUC)
- Hyperparameter tuning using Cross Validation.
- Saving the models

Planned Analysis:

The most popular metric for a binary classification problem is *accuracy*. Other methods of evaluating probabilities include, using area under the ROC Curve as well as the F1 metric. The AUC provides an aggregate measure of performance. The AUC is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve. The higher the AUC, the better the performance of the model to distinguish between positive and negative classes.

A *Confusion Matrix* summarizes the results of a binary classifier, including sensitivity (TPR) and specificity (TNR), Precision (Positive Predictive Value) and error rate (ratio of instances misclassified), True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN).

Accuracy = TP + TN / TP + FP + TN + FN or (1 – Error Rate). Accuracy is the correct predictions divided by the total predictions.

Accuracy fails on classification problems with severely skewed class distribution. If a severely unbalanced dataset is used, then a Synthetic Minority Oversampling Technique (SMOTE) will be necessary. A dataset that is severely imbalanced is one that has a majority-to-minority class ratios between 100:1 and 10,000:1. This data, although unbalanced is not severely so. The income dataset was 75:25, so was not considered to be a candidate.

Error Rate = Incorrect predictions / total predictions or (1 – accuracy)

F1 Score is used to determine a good model as it combines Precision and Recall obtaining a balanced classification model.

Area under ROC curve (AUC): The receiver operating characteristic curve is one of the most useful testing analysis methods in binary classification. It provides a visual way of summarizing the accuracy of a classifier. False positive rate values are plotted on the x-axis and corresponding Sensitivity (TPR) are plotted on the y-axis. The closer to the point of (0,1) the better the classifier. The AUC is a better measure of performance as it doesn't bias on size of test or evaluation data. AUC also considers all possible thresholds. It is therefore often preferred rather than accuracy. Because data is often imbalanced, this imbalance has a large effect on the Positive Rate (TP) but this is not the case with ROC/AUC as it is insensitive to class imbalance. There is however a lot of debate about which measure is best. As such the results of the modelling on the Income dataset include accuracy, F1 Score and ROC.

Summary and Conclusions

The following results were obtained from the modelling processes:

	Area Under ROC Curve	F1	Accuracy
Logistic Regression	0.76	0.84	0.84
Logistic Regression (with Cross Validation)	0.77	0.84	0.85
Decision Tree	0.71	0.82	0.78
Naïve Bayes (with Cross Validation)	0.59	0.84	0.85
Random Forest	0.71	0.82	0.84
Support Vector Machine	0.75	0.84	0.84
Gradient Boost	0.75	0.84	0.85

Based on the results listed above, the model with the highest area under ROC is the Logistic Regression with Cross Validation. Support Vector Machine (SVM), Gradient Boost and Logistic Regression all performed well in predicting income. It is possible that either the Decision tree, Random Forest, Support Vector Machine or Gradient Boost methods would have had higher ROC values if the Cross Validation models were able to be run. However, this was not possible, probably due to a strain in computing resources on the UOL Cluster. Because the data was relatively imbalanced, results could have also been improved by running a SMOTE analysis.

The hypotheses that income can be accurately predicted using supervised machine learning techniques and a binary classification methodology is confirmed.

Files:

Three .py files are included as part of this project. One of the .py files was used to copy and paste directly into the Pyspark terminal. A second file is a script file that can be run with spark-submit in the terminal window. The third file is one that has been formatted to run on the Jupyter Hub. Access to the Jupyter Hub was not possible due to some technical issues.

How challenges were addressed

Because of the problem accessing the Jupyter Hub on Lena, the commands were copied and pasted from a local Python (.py) file directly into the PySpark Terminal. The file was then used to format a script file that can be run using spark-submit. Because it was not possible to utilize the Jupyter Hub, no graphs were created, although a formatted script file was created, it was never tested. All commands were run directly in the terminal.

Because running Cross Validation models for Decision tree, Random Forest, Support Vector Machines and Gradient Boost models took up too much computing resources, the associated code was hashed out, but left in the .py files.

Application to larger datasets

Applying these methods to larger datasets is dependent on several factors, including:

- Working with smaller samples until a final model is fitted on all the data.
- Using computers with more memory, for example renting time on a cloud service like Amazon Web Services. AWS offers high RAM at relatively affordable rates.
- Changing the data format to a binary one could increase the speed of data loading and memory usage.
- Streaming data will reduce the need to use as much memory for training. Algorithms that learn iteratively using optimization techniques work better than those loading all data into memory. An example of this is stochastic gradient descent methods.
- Progressive loading of image files is available in the Keras deep learning library. The Pandas library is also capable of loading large CSV files in chunks.
- Avoid underfitting or overfitting the dataset. To avoid oversampling, more training data would be needed or choosing a model with fewer features. Underfitting can be reduced by better feature engineering, increasing parameters, and removing noise from the dataset.

2. Implementation of the project

Techniques and Methodology

1. Create a Spark Session and read in the data:

The PySpark API was used to write the Python code. A spark session is initialized by passing an Application name “income” to a string, to *.appName()* as an argument. The *.getOrCreate()* creates the *SparkSession*. The Spark *DataFrame* was created from an external .csv file. This file was uploaded to the Hadoop Distributed File System (HDFS) on the Lena cluster (Goldsmiths). The *printSchema()* method displays the columns and their data type. Fig 1 shows this schema.

```
>>> df.printSchema()
root
 |-- age: integer (nullable = true)
 |-- workclass: string (nullable = true)
 |-- education: string (nullable = true)
 |-- educational-num: integer (nullable = true)
 |-- marital-status: string (nullable = true)
 |-- occupation: string (nullable = true)
 |-- relationship: string (nullable = true)
 |-- race: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- capital-gain: integer (nullable = true)
 |-- capital-loss: integer (nullable = true)
 |-- hours-per-week: integer (nullable = true)
 |-- native-country: string (nullable = true)
 |-- income: string (nullable = true)
```

Fig 17 Schema of data

2. Data pre-processing:

2.1 Dropping features: the "fnlwgt" feature column refers to population totals and is created by weighted summaries. The "x" feature is an ID column. Both "x" and "fnlwgt" features are dropped from the dataframe. A *printSchema* of the resulting DataFrame is shown below:

```
>>> df.printSchema()
root
 |-- age: integer (nullable = true)
 |-- workclass: string (nullable = true)
 |-- education: string (nullable = true)
 |-- educational-num: integer (nullable = true)
 |-- marital-status: string (nullable = true)
 |-- occupation: string (nullable = true)
 |-- relationship: string (nullable = true)
 |-- race: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- capital-gain: integer (nullable = true)
 |-- capital-loss: integer (nullable = true)
 |-- hours-per-week: integer (nullable = true)
 |-- native-country: string (nullable = true)
 |-- income: string (nullable = true)
```

Fig. 18 Schema of data after dropping features

2.2 Cast integer columns to float: The integer columns "age","capital-gain","capital-loss" and "hours-per-week" are cast to float.

```
>>> df_cast.printSchema()
root
 |-- workclass: string (nullable = true)
 |-- education: string (nullable = true)
 |-- marital-status: string (nullable = true)
 |-- occupation: string (nullable = true)
 |-- relationship: string (nullable = true)
 |-- race: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- native-country: string (nullable = true)
 |-- income: string (nullable = true)
 |-- age: float (nullable = true)
 |-- capital-gain: float (nullable = true)
 |-- educational-num: float (nullable = true)
 |-- capital-loss: float (nullable = true)
```

Fig. 19 printSchema after dropping features

2.3 Missing values: There are a few missing values in the categorical features, these appear within, workclass, occupation and native-country. A default symbol of "?" has been assigned. The "?" have been replaced by "None".

```
workclass         with '?' values:  2799
education         with '?' values:  0
marital-status    with '?' values:  0
occupation        with '?' values:  2809
relationship      with '?' values:  0
race      with '?' values:  0
gender    with '?' values:  0
native-country    with '?' values:  857
income    with '?' values:  0
age       with '?' values:  0
capital-gain      with '?' values:  0
educational-num           with '?' values:  0
capital-loss      with '?' values:  0
hours-per-week    with '?' values:  0
```

Fig. 20 Missing values by feature

```
workclass         with null values:  0
education         with null values:  0
marital-status    with null values:  0
occupation        with null values:  0
relationship      with null values:  0
race      with null values:  0
gender    with null values:  0
native-country    with null values:  0
income    with null values:  0
age       with null values:  0
capital-gain      with null values:  0
educational-num           with null values:  0
capital-loss      with null values:  0
hours-per-week    with null values:  0
```

Fig. 21 After removal of missing values

2.4 Adding a new feature column: The "age-squared" is a calculated column to allows for modelling the effect a differing age has rather than assuming the effect is linear for all ages.

```
root
 |-- workclass: string (nullable = true)
 |-- education: string (nullable = true)
 |-- marital-status: string (nullable = true)
 |-- occupation: string (nullable = true)
 |-- relationship: string (nullable = true)
 |-- race: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- native-country: string (nullable = true)
 |-- income: string (nullable = true)
 |-- age: float (nullable = true)
 |-- capital-gain: float (nullable = true)
 |-- educational-num: float (nullable = true)
 |-- capital-loss: float (nullable = true)
 |-- hours-per-week: float (nullable = true)
 |-- age-squared: double (nullable = true)
```

Fig. 22 Addition of age-squared feature

2.5 Dropping values: only 1 row of data was available for the native-country, Holand-Netherlands, this row was dropped as it added no value to the output.

After dropping the null values, removing the single record for Holand-Netherlands and adding the new feature column "age-squared", the dataframe now has a total of 15 columns and 45221 rows.

2.6 Class counts:

```
#>>> class_counts_dict
{'<=50K': 34013, '>50K': 11208}
```

Fig. 23 Class count

2.7 Statistical analysis: The dataframe was converted to a pandas dataframe to do the statistical analysis.

```
>>> print(df_numeric.describe())
                age  educational-num  ...  hours-per-week   age-squared
count  47984.000000     47984.000000  ...    47984.000000  47984.000000
mean      38.641422        10.067231  ...       40.411491   1681.661095
std       13.729731         2.560561  ...       12.391202   1188.496055
min       17.000000         1.000000  ...        1.000000    289.000000
25%       28.000000         9.000000  ...       40.000000    784.000000
50%       37.000000        10.000000  ...       40.000000   1369.000000
75%       48.000000        12.000000  ...       45.000000   2304.000000
max       90.000000        16.000000  ...       99.000000   8100.000000
```

Fig. 24 Descriptive statistics by feature

2.8 Encoding of Categorical features:

Encoding was done in two stages.

a. Label Encoding:

All categorical features are label encoded. This was achieved with *StringIndexer*.

b. One-Hot Encoding:

The One-Hot Encoding method splits the categorical features into separate categories where each category obtains a binary value of 0 or 1. 0 if it belongs and 1 if it does not. *OneHotEncoder* was used from the Spark ml library.

2.9 Vector Assembling:

The *VectorAssembler* transforms a list of columns into a single vector column, The Vector Assembler can accept various input column types, such as numeric, Boolean and vector. The values in each row of the input columns are concatenated into an order-specified vector.

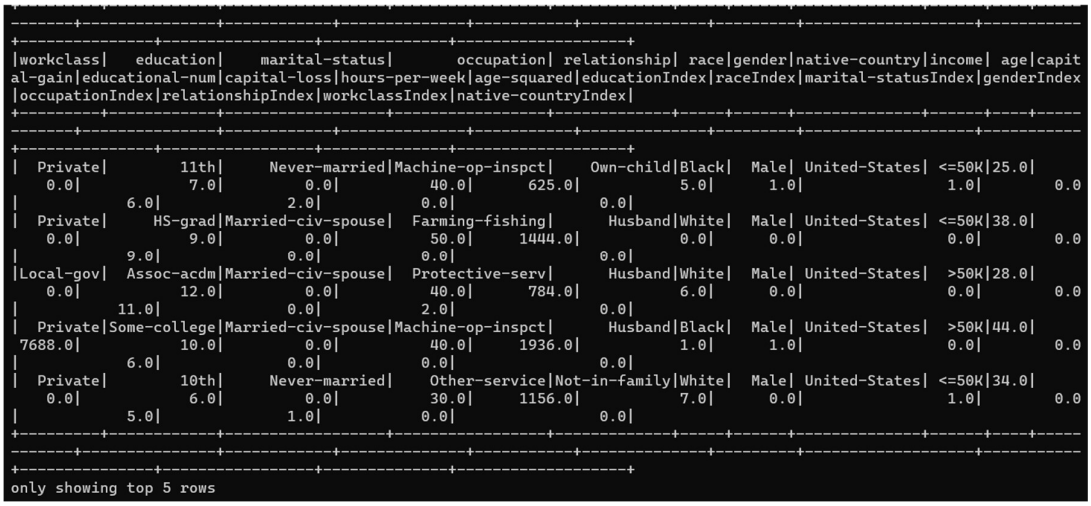


Fig. 25 Vector Assembler

2.10 Preparing training and testing data frames:

After pre-processing the data, it was split randomly into training and testing datasets with an 80% / 20% random split. For replication purposes, the seed was set to 13.

3. Modelling the data

Pipelines:

A Pipeline is a high-level API for MLlib. The Spark Pipeline is a sequence of stages, either Transformer or Estimator that are run in order and the DataFrame is transformed as it is passed through each stage. The Pipline was fitted to each model with the following stages: *stringIndexer, encoder, labelToIndex, assembler* and *model.* The *.fit()* function was run on the training data for each pipeline model. The *.transform()* function was applied to the testing data for each model.

A total of eight Pipeline models were created and saved onto the HDFS server.

3.1 Logistic Regression:

Logistic Regression is a statistical predicting model which predicts either 0 or 1 using the Sigmoid function. If the numeric data is put into this model, the output will either be < 0.5 which will be classified as 0 or > 0.5 in which case a classification of 1 will be assigned. The binomial logistic regression method was used to predict the binary outcome and the multiclass prediction was achieved by using multinomial logistic regression.

Predictions and probability:

```
>>> lr_select = lr_predict.select("label", "prediction", "probability").show(5)
+-----+----------+--------------------+
|label|prediction|         probability|
+-----+----------+--------------------+
|  0.0|       0.0|[0.96106400019659...|
|  0.0|       0.0|[0.77176743074186...|
|  0.0|       0.0|[0.96687519083902...|
|  0.0|       0.0|[0.79265165688381...|
|  1.0|       0.0|[0.55044529797630...|
+-----+----------+--------------------+
only showing top 5 rows
```

Fig. 26 Prediction and probability for lr pipeline model

```
>>> # Confusion Matrix
>>> lr_cm_predict = lr_predict.crosstab("prediction", "label").show()
+-----------------+----+----+
|prediction_label| 0.0| 1.0|
+-----------------+----+----+
|              1.0| 472|1271|
|              0.0|6196| 916|
+-----------------+----+----+
```

Fig. 27 Confusion matrix for lr model

Hyper Parameter Tuning – Cross Validation

Hyper-parameter tuning (Cross Validation) was made to the Logistic Regression model. ROC and accuracy improved compared with the original model, accuracy was improved by 1% to 0.85 and ROC by 1% to 0.77. The F1 score remained the same.

```
>>> lr_cv_select = lr_cv_predict.select("label","prediction","probability").show(5)
+-----+----------+--------------------+
|label|prediction|         probability|
+-----+----------+--------------------+
|  0.0|       0.0|[0.97148444521629...|
|  0.0|       0.0|[0.74284703235287...|
|  0.0|       0.0|[0.97766724780275...|
|  0.0|       0.0|[0.76468145857651...|
|  1.0|       1.0|[0.48556800249242...|
+-----+----------+--------------------+
only showing top 5 rows
```

Fig. 28 Lr CV predictions and probability

3.2 Naïve Bayes

The Naïve Bayes algorithm is based on Bayes theorem. It's mainly used in text classification of high-dimensional data. It predicts based on the probability of an object. It works well on spam filtration and sentiment analysis. Although it is fast and easy, it assumes features are independent so can't learn this relationship. This algorithm is not as effective as others for this dataset because of this limitation. The area under ROC is only 58% although accuracy and F1 measures are higher, 78% and 72% respectively. The ROC measure is a more reliable one than either accuracy or F1.

```
>>> nb_select = nb_predict.select("label","prediction","probability").show(5)
+-----+----------+-----------+
|label|prediction|probability|
+-----+----------+-----------+
|  0.0|       0.0|  [1.0,0.0]|
|  0.0|       0.0|  [1.0,0.0]|
|  0.0|       0.0|  [1.0,0.0]|
|  0.0|       0.0|  [1.0,0.0]|
|  1.0|       0.0|  [1.0,0.0]|
+-----+----------+-----------+
only showing top 5 rows
```

Fig. 29 Predictions and probabilities Naïve Bayes

```
>>> nb_predictions.groupBy("label","prediction").count().show()
+-----+----------+-----+
|label|prediction|count|
+-----+----------+-----+
|  1.0|       1.0|  461|
|  0.0|       1.0|  254|
|  1.0|       0.0| 1726|
|  0.0|       0.0| 6414|
+-----+----------+-----+
```

Fig. 30 Naïve Bayes prediction count

3.3 Decision Tree

This algorithm will find the most significant independent variables to create a group; it makes decisions using a tree-based model. It looks at all the data features, then decides on the one with the highest accuracy, then binary split is performed, and it repeats until it successfully splits the data in all the leaves or reaches the maximum depth. The GINI index measures the amount of probability of a specific feature that is classified incorrectly when selected randomly.

```
>>> dt_predict.select("label","prediction","probability").show(5)
+-----+----------+--------------------+
|label|prediction|         probability|
+-----+----------+--------------------+
|  0.0|       0.0|[0.94869510533487...|
|  0.0|       0.0|[0.67991704113377...|
|  0.0|       0.0|[0.94869510533487...|
|  0.0|       0.0|[0.67991704113377...|
|  1.0|       0.0|[0.67991704113377...|
+-----+----------+--------------------+
only showing top 5 rows
```

Fig. 31 Decision Tree predictions and probabilities

```
>>> # Confusion Matrix
>>> dt_cm_predict = dt_predict.crosstab("prediction",\
...                                     "label").show()
+----------------+----+----+
|prediction_label| 0.0| 1.0|
+----------------+----+----+
|             1.0| 352|1030|
|             0.0|6316|1157|
+----------------+----+----+
```

Fig. 32 Decision Tree Confusion Matrix

3.4 Random Forest

The Random Forest algorithm builds decision trees and takes the majority vote in a classification problem. One of its most appealing advantages is the fact that it capable of handling continuous and categorical features. It is an ensemble method that combines many classifiers solving complex problems. It's trained through bagging and bootstrap aggregating. The more trees one uses, the higher the precision. Random Forest performs better than the Decision Tree algorithm as it eliminates overfitting and increases precision and accuracy at the same time. It also performs well without hyper-parameter tuning.

```
>>> rf_predict.select("label","prediction","probability").show(5)
+-----+----------+--------------------+
|label|prediction|         probability|
+-----+----------+--------------------+
|  0.0|       0.0|[0.88867548588989...|
|  0.0|       0.0|[0.68053378024324...|
|  0.0|       0.0|[0.91631959866473...|
|  0.0|       0.0|[0.67922473086084...|
|  1.0|       0.0|[0.63986782751412...|
+-----+----------+--------------------+
only showing top 5 rows
```

Fig. 33 Random Forest predictions and probabilities

```
>>> rf_cm_predict = rf_predict.crosstab("prediction", "label").show()
+----------------+----+----+
|prediction_label| 0.0| 1.0|
+----------------+----+----+
|             1.0| 301|1021|
|             0.0|6367|1166|
+----------------+----+----+
```

Fig. 34 Random Forest Confusion Matrix

3.5 Support Vector Machines (SVMs)

SVMs are best suited for classification problems. The main aim of the algorithm is to find a hyperplane in an N-dimensional space to classify the data points distinctly. The dimension of the hyperplane depends on how many features are contained within a dataset. A dataset with three features has a 2-D plane. SVM ignores outliers. The SVM kernel is a function that takes low dimensional input space and transforms it into higher-dimensional space, which is useful in non-linear separation problems. It can do extremely complex data transformations. SVM has the advantage of efficiency in high dimensional data, uses memory well because it uses a subset of training points called support vectors; different kernel functions can be specified, they can also be customized.

```
>>> svm_predict.select("label","prediction").show(5)
roupBy("label","prediction").count().show()
+-----+----------+
|label|prediction|
+-----+----------+
|  0.0|       0.0|
|  0.0|       0.0|
|  0.0|       0.0|
|  0.0|       0.0|
|  1.0|       0.0|
+-----+----------+
only showing top 5 rows
```

Fig. 35 SVM prediction and probabilities

```
>>> svm_predict.groupBy("label","prediction").count().show()
+-----+----------+-----+
|label|prediction|count|
+-----+----------+-----+
|  1.0|       1.0| 1224|
|  0.0|       1.0|  423|
|  1.0|       0.0|  963|
|  0.0|       0.0| 6245|
+-----+----------+-----+
```

Fig. 36 SVM prediction count

```
>>> svm_cm_predict = svm_predict.crosstab("prediction","label").show()
+----------------+----+----+
|prediction_label| 0.0| 1.0|
+----------------+----+----+
|             1.0| 423|1224|
|             0.0|6245| 963|
+----------------+----+----+
```

Fig. 37 SVM Confusion Matrix

3.6 Gradient Boost

Gradient boosting is also an ensemble method of learning, whereby weak learners are altered to become better. The algorithm minimizes a loss function by iteratively choosing a function that points towards the negative gradient. A loss function is one that estimates how good a model is at making predictions.

```
>>> gbt_predict.select("label","prediction","probability").show(5)
"label","prediction").count().show()
+-----+----------+--------------------+
|label|prediction|         probability|
+-----+----------+--------------------+
|  0.0|       0.0|[0.91546134925305...|
|  0.0|       0.0|[0.86848372589918...|
|  0.0|       0.0|[0.92335557766575...|
|  0.0|       0.0|[0.86848372589918...|
|  1.0|       0.0|[0.78794546427275...|
+-----+----------+--------------------+
only showing top 5 rows
```

Fig. 38 Gradient Boost predictions and probabilities

```
>>> # Confusion Matrix
>>> gbt_cm_predict = gbt_predict.crosstab("prediction","label").show()
+----------------+----+----+
|prediction_label| 0.0| 1.0|
+----------------+----+----+
|             1.0| 379|1203|
|             0.0|6289| 984|
+----------------+----+----+
```

Fig. 39 Gradient Boost Confusion Matrix

Bibliography

Brownlee, J. (2020, December 10). *github.com*. Retrieved from Machine Learning Datasets: https://github.com/jbrownlee/Datasets

Brownless, J. (2017, May 29). *machinelearningmastery*. Retrieved from 7 ways to handle large data files for machine learning: https://machinelearningmastery.com/large-data-files-machine-learning/

Chen, J. (n.d.). *Wisconsin edu*. Retrieved from Feature Significance Analysis of the US Adult Income Dataset: https://minds.wisconsin.edu/bitstream/handle/1793/82299/TR1869%20Junda%20Chen%203.pdf?sequence=1&isAllowed=y

Deepajothi, S., & Selvarajan, D. S. (2012, October). A Comparative Study of Classification Techniques On Adult Data Set. *International Journal of Engineering Research & Technology, 1*(8). Retrieved from https://www.ijert.org/research/a-comparative-study-of-classification-techniques-on-adult-data-set-IJERTV1IS8243.pdf

GeeksforGeeks. (2021, January 22). *GeeksforGeeks*. Retrieved from Support Vector Machine Algorithm: https://www.geeksforgeeks.org/support-vector-machine-algorithm/

honzaMaly. (2020, January 8). *github.com*. Retrieved from census_income_data_pyspark_classification: https://github.com/honzaMaly/census_income_data_pyspark_classification

hoytlui. (2021, August 17). *github.com*. Retrieved from Data Science Case Studies: https://github.com/hoytlui/DataScienceCaseStudies

JavaTPoint. (n.d.). *JavaTPoint*. Retrieved from Naive Bayes Classifier Algorithm: https://www.javatpoint.com/machine-learning-naive-bayes-classifier

Johnson, D. (2022, March 8). *Guru99*. Retrieved from Pyspark Tutorial for Beginners: Learn with EXAMPLES: PySpark Tutorial for Beginners: Learn with EXAMPLES (guru99.com)

Kurama, V. (2020). *paperspace blog*. Retrieved from Gradient Boosting in Classification: Not a Black Box Anymore : https://blog.paperspace.com/gradient-boosting-for-classification/

Leevy, J. L., Khoshgoftaar, T. M., Bauder, R. A., & Seliya, N. (2018). A survey on Addressing high-class imbalance in big data. *Journal of Big Data*. Retrieved from https://journalofbigdata.springeropen.com/articles/10.1186/s40537-018-0151-6

Li, S. (2018, May 6). *Towards Data Science*. Retrieved from Machine Learning with Pyspark and MLlib - Solving a Binary Classification Problem: https://medium.com/towards-data-science/machine-learning-with-pyspark-and-mllib-solving-a-binary-classification-problem-96396065d2aa

Mbaabu, O. (2020, December 11). *Section*. Retrieved from Introduction to Random Forest in Machine Learning: https://www.section.io/engineering-education/introduction-to-random-forest-in-machine-learning/

Petrushev, A. (2021, August 3). *github.com*. Retrieved from Predicting income class using Pyspark Machine Learning: https://github.com/aleksandarpetrushev/pyspark-data-modeling

rutvikdixit. (2019, December 11). *github.com*. Retrieved from Big Data Census Income using AWS EMR, S3, Pyspark, Pig and Hive: https://github.com/rutvikdixit/BigDataCensusIncome

Shrsh. (2018, May 28). *github.com*. Retrieved from Random Forests on Income classification: Shrsh/UCI--Adult-Data-Set: Random Forests on Income classification (github.com)