Informing Education Policy: Predicting Education Level Using Census Data

DS5110: Big Data Systems (Summer 2021) - Final Project Report

Ed Squad: Shilpa Narayan (smn7ba), Jamie Oh (hso6b), Ashlie Ossege (ajo5fs), Isaac Stevens (is3sb)

Abstract

To implement effective education reform across the United States, it is vital to know areas where education programs would be most effective. These areas could be identified as areas where higher levels of education are rarely attained. This paper uses five years of detailed census data from Ipums.org to predict an individual's education level. Through utilizing big data analysis techniques such as Principal Component Analysis (PCA), Cross Validation, and Grid Search, we trained a Random Forest model with high accuracy (80.2%) and recall (76.4%) that effectively predicts education level while minimizing False Negatives (individuals predicted to be uneducated, but actually are educated). Several models were trained and evaluated, such as Logistic Regression, Support Vector Machine, and Gradient Boosting, however the Random Forest yielded the highest accuracy and recall, which would be best to maximize identifying areas that are truly uneducated so they can receive education assistance.

Introduction

Inspired by the question "How much has your ZIP Code Determined Your Opportunities," posed in the Student Opinion section of NY Times[1]. The Ed Squad seeks to answer the question:

What factors of a household contribute to completing education?

Specifically, we hypothesize that Public Use Microdata Areas (PUMA)[2] will be one of the leading indicators of whether one is predicted to complete their education. Our census data source from Ipums.org does not contain ZIP code identifiers in public-use data, which is why we are looking at the most granular geographic factor our dataset has available.

Why is this important?

When regions can be identified as being more or less likely to complete a higher education, public and private programs and budgets can be developed, and resources allocated objectively where they are needed the most.

Data and Methods

Data

Our data comes from the American Community Survey 2015-2019, 5-Year Sample from usa.ipums.org. and contains all households and persons from 1% ACS sample for 2015-2019 identifiable by year. The data includes information such as age, income, health insurance, and other demographic variables. In total, there are 137 variables, and the file is 2.3GB. Our focus of the study is on the South Region. The smallest identifiable geographic unit is the PUMA. PUMAs contain at least 100,000 people.

Data: https://usa.ipums.org/usa/sampdesc.shtml#us2019c
Code Book for variables in the data: https://github.com/narashil/DS-5110-Project/blob/main/usa_00001.cbk

Supplemental Sources:
[1] https://www.nytimes.com/2020/05/19/learning/how-much-has-your-zip-code-determined-your-opportunities.html
[2] https://www.census.gov/programs-surveys/geography/guidance/geo-areas/pumas.html

Exploratory Data Analysis

In the initial exploratory data analysis, we checked the shape (number of rows and columns) of the dataframe that stores all the data read from the source file. Our response variable was *"EDUC"*, which gave the last level of education completed by a person. To simplify, we created a binary label to identify if someone is educated beyond grade 12 or not.

After creating the label, we performed *groupby* transformations, then converted the data to a pandas dataframe for visualizations. Various bar plots give more context to the data and identify any other imbalances.

Education level counts on our dataset are show in Figure 1.

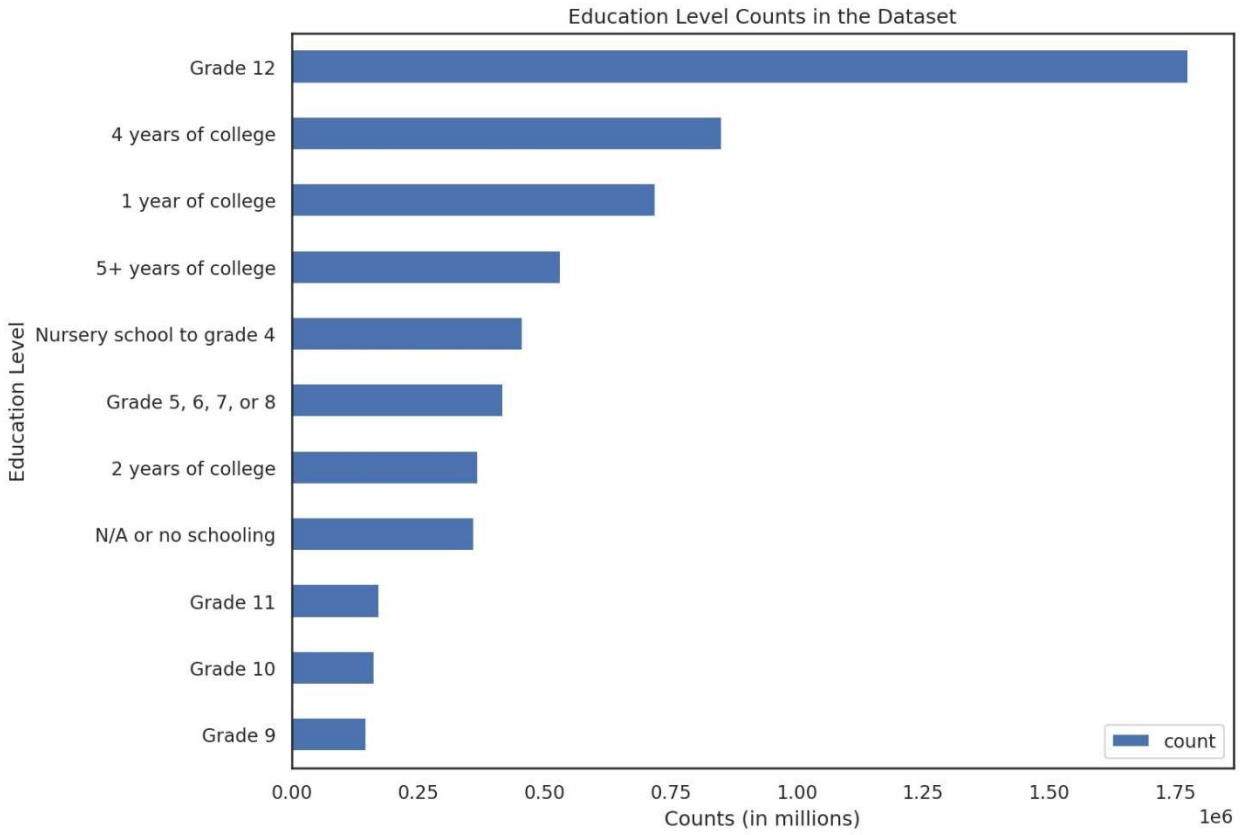

Figure 1: Education level counts

Counts by gender, are shown in Figure 2. For males, 60.61% are flagged as not educated and 39.39% are educated. On the other hand, 56.69% of females are flagged as not educated and 43.31% as educated.

We visualized more bar plots such as household distributions in our data. Most of the households were married. Those plots can be found in the code notebook.

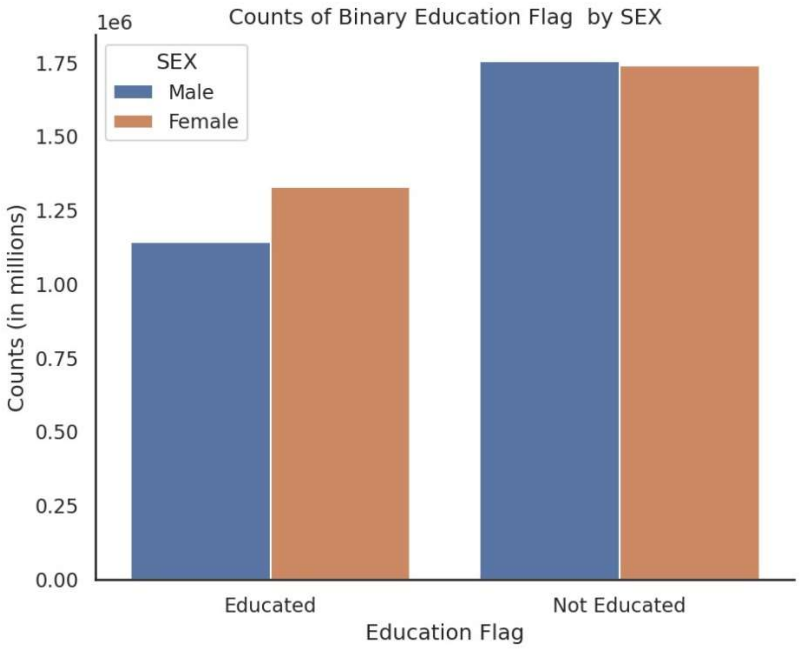

Figure 2: Counts by gender and education flag

Label

We created a binary classification of our response variable "EDUC" where observations that had completed education past high school were classified as 1, and all others as 0. We did sanity checks to make sure the label was created accurately. Our label variable was unbalanced, so we downsampled to balance the label.

Before balancing:

```
+-----+-------+
|label|  count|
+-----+-------+
|  0.0|3495122|
|  1.0|2470127|
+-----+-------+
```

After balancing:

```
+-----+-------+
|label|  count|
+-----+-------+
|  0.0|2471011|
|  1.0|2470127|
+-----+-------+
```

Figure 3: Counts of education flag before and after downsampling

Sampling

We sampled 10% of the balanced data by each of the five years of data for better performance.

Models

The approach to this project was to build four models.

1. Logistic Regression: A basic model used as a benchmark.
2. Support Vector Machines (SVM): Used for accuracy and less computational power
3. Gradient Boosting: Sequence of tree building and combining predictions at each iteration
4. Random Forest: Ensembles of decision trees, feature selection and feature importance

Feature Selection & Engineering

We removed variables that were not relevant to our analysis such as household weight, serial number, and variable descriptions. We do not have the specific domain of knowledge regarding the prediction of education; however, we hypothesized which of the 137 features may most likely predict education. Therefore, we chose 36 features for our benchmark model. A summary of some of these selected features are as follows:

- Marital Status
- Geographic Indicators (state, county, puma, metro)
- Cost of utilities (electric, gas, water, fuel)
- FoodStamp
- Presence of Technology (cell phone, laptop, internet, tablet)
- Family Composition (mothers, fathers, coupletype)
- Race Flags (Am.Indian, Asian, Black, Pacific Islander, White, Other)
- Employment Status
- Veteran Status
- Transportation (Vehicles Owned, Work Transportation)

Our basic model using logistic regression on these 36 variables gave approximately 68% accuracy.

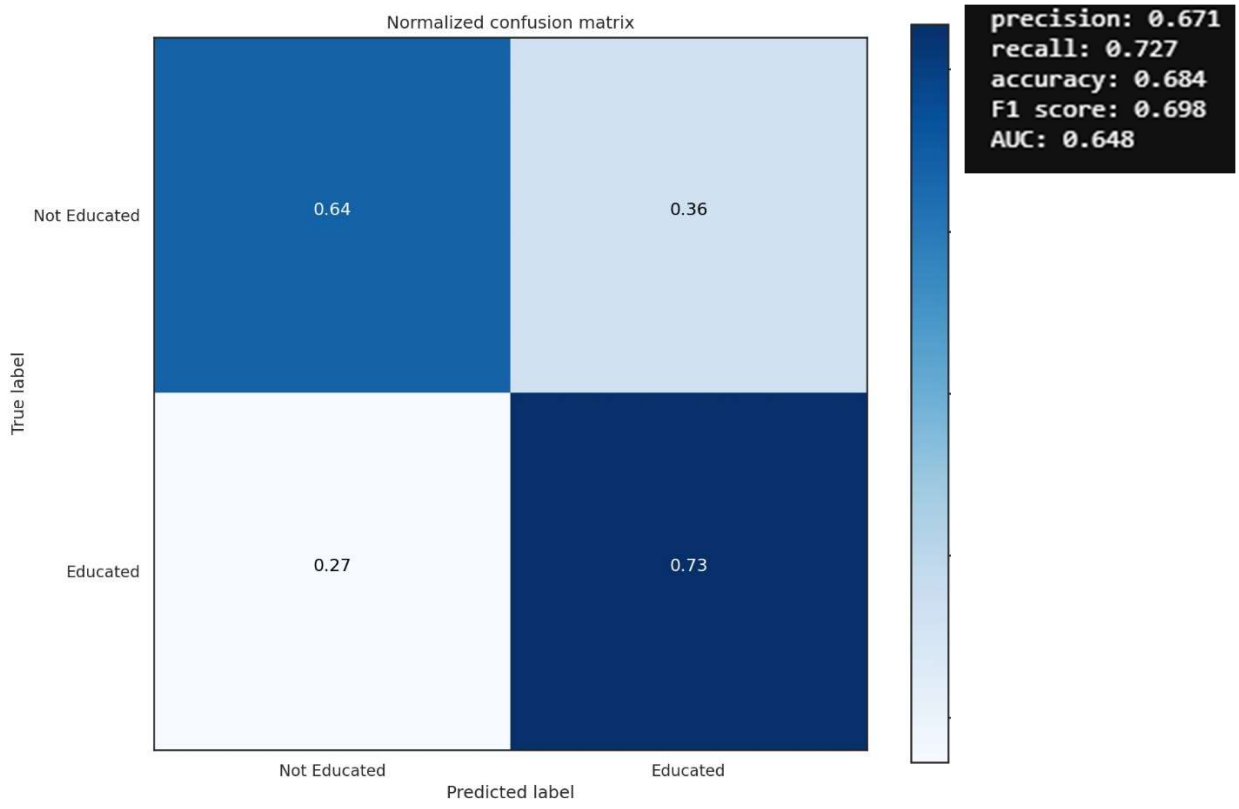

Figure 4: Confusion Matrix and classification metrics for baseline logistic model

To improve accuracy but still reduce dimensions, we applied Principal Component Analysis (PCA) for the remaining models, which we determined is a better approach for more automated reduction. Note, PCA is explained in detail under the Dimension Reduction section.

Since PCA tries to cluster all the variables based on the variance, we wanted to make sure we remove any highly correlated variables which did not add value to the models. Therefore, we created a correlation matrix by converting a PySpark dataframe into a pandas dataframe and using the correlation function to visualize the matrix.

Correlation Matrix

A correlation matrix on the data was generated to help identify unnecessary variables. For example, we identified various income fields that were highly correlated with the total income field and were not adding any additional value. We removed such fields to keep the principal components clean. Finally, the count of variables dropped to 106.

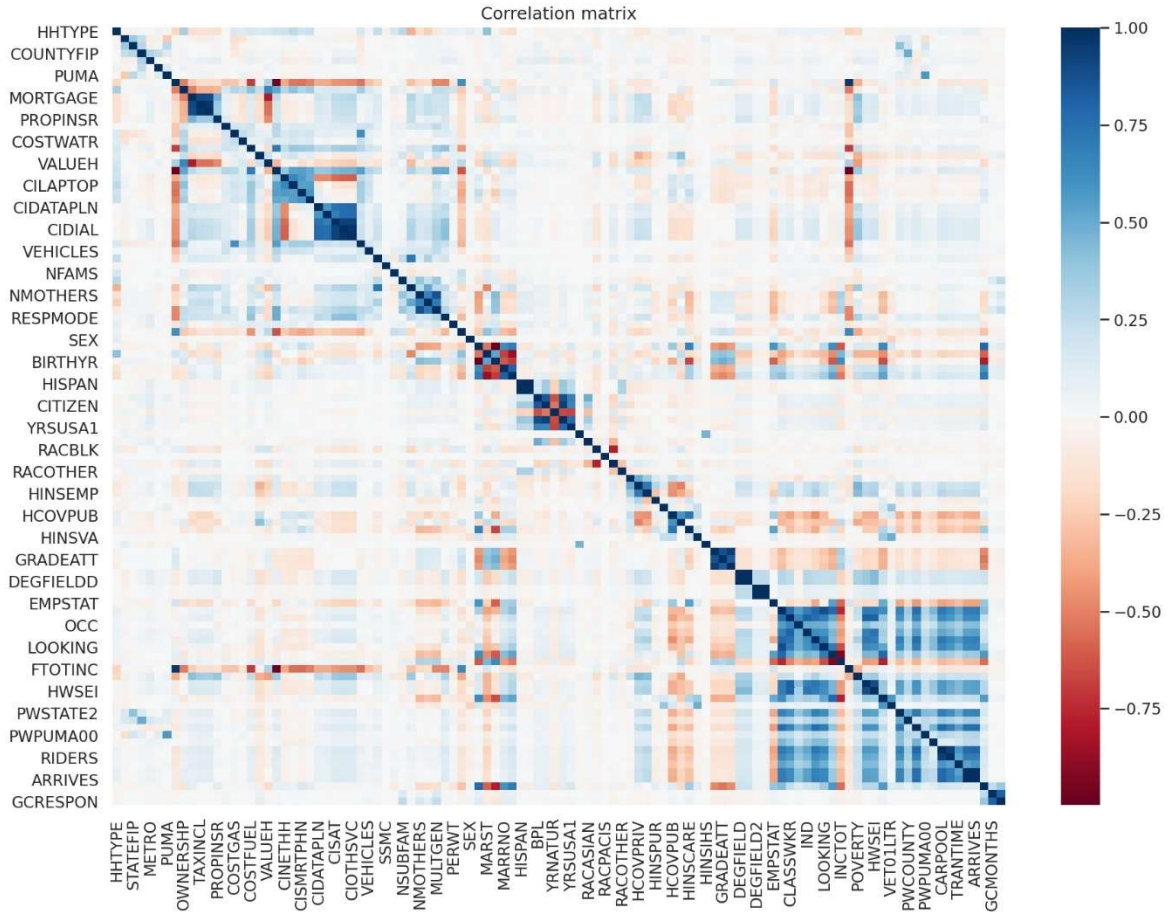

Figure 5: Correlation matrix for the independent variables in the dataset

Scaling

Generally, it is a good idea to scale data because if there is a vast difference in the range, for example few ranging in thousands and few ranging in the tens, the model can make the underlying assumption that higher ranging numbers may be more significant. The machine learning algorithms work on numbers and do not know what that number represents. A weight of 10 grams and a price of 10 dollars represents completely two different things but for a model as a feature, it treats both as the same. Therefore, it is a good practice to scale the data unless we are doing language models.

We scaled our training data after splitting the data to prevent the data leakage or information leakage which can overfit the models.

Another important point to note while scaling is that the tree classifiers do not need the scaled data like other algorithms. However, to keep our preprocessing steps consistent and to write some reusable code, we went ahead and scaled the data and used it consistently in all the models. The one great advantage is that having such reusable code can let us experiment with other models in the future.

Dimension Reduction

PCA is a primary technique used for dimension reduction. PCA constructs a set of vectors that are a linear combination of the original vectors. One of the benefits of using PCA is that our features (vectors) will be uncorrelated. In addition, each component contains a portion of the variability of the data, with the first component accounting for the most, and the second accounting for the second most, and so on.

Interpreting Principal Components

The scree plot showed us that the first principal component accounts for almost 14% of the variance, and we saw a sudden dip in the rest of the components. Therefore, we can certainly interpret the features contained in the first principal component as more significant in predicting if a person is educated or not in our dataset.

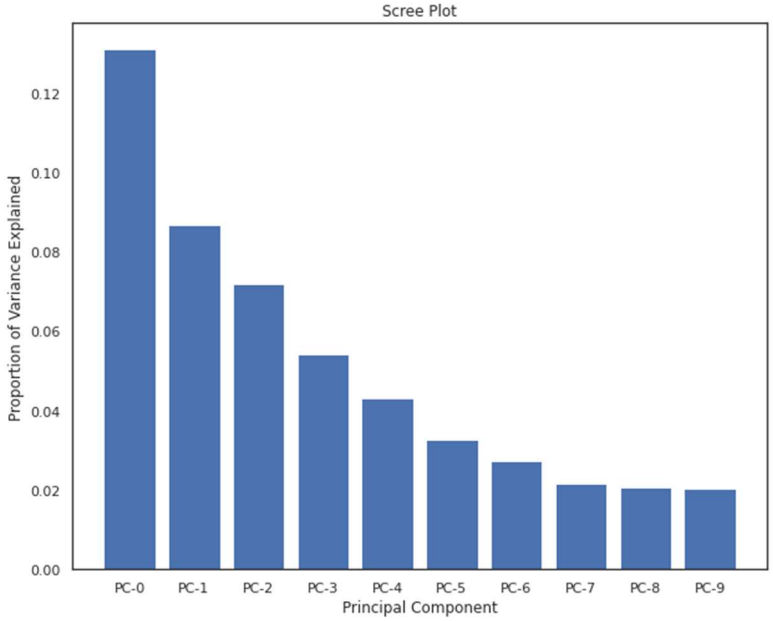

Figure 6: The scree plot showing the explained variance for each principal component

We further explored the components by plotting the features as index for the loadings of various principal components. This provided great insight into the features which are important. We also visualized the scatter plot using *plotly express* interactive graphs to see how the features are clustered in the components.

```
PC0+ LABFORCE UHRSWORK CLASSWKR HWSEI OCCSCORE LOOKING PWTYPE CARPOOL
PC0- INCTOT MARST HINSCAID HCOVPUB SCHOOL RELATE SCHLTYPE GRADEATT
PC1+ AGE FTOTINC GQ EMPSTAT VETSTAT AVAILBLE HINSCARE GCHOUSE
PC1- MULTGEN FRIDGE BIRTHYR CIDIAL CIOTHSVC CISAT INCTOT NMOTHERS
```

Figure 7: The top 8 and bottom 8 variables based on positive and negative loadings of first two components

Reducing the number of dimensions of a data can reduce the accuracy, but always a simpler model is more interpretable and preferred over complex ones. Being able to see which features contribute to the components is more meaningful and can be explained easily to other interested stakeholders.

Hyperparameter Tuning

To improve the model(s) predictive power, we defined the hyperparameters. A hyperparameter is an attribute that you define about the model prior to training, and it is not learned during the training process. The number of trees in our classification models is an example of a hyperparameter.

```
#SVM param grid
paramGrid_svm = ParamGridBuilder() \
    .addGrid(pca_model.k,[10,20,30])\
    .addGrid(lsvc.regParam, [0.1,0.5,1]) \
    .build()

#Gradient param grid
paramGrid_gbt = ParamGridBuilder() \
             .addGrid(gb.maxIter, [10, 15]) \
             .addGrid(gb.maxBins, [10, 20]) \
             .build()

#Random Forest param grid
paramGrid_rf = ParamGridBuilder() \
        .addGrid(rf.numTrees, [20, 30]) \
        .build()
```

Figure 8: Parameter grids for each model

Cross Validation

We used k-fold cross validation for model performance evaluation by applying the *CrossValidator* method, instead of splitting the dataset into a single training and test set (*TrainValidationSplit*). One of the benefits of cross-validation is reducing the possibility of over-fitting with a certain train/test split. It can also increase the model's accuracy by finding the best model, though it can be computationally expensive.

We chose different hyperparameters for different models as discussed above and created separate parameter grids which we passed the models, evaluator, and the number of folds to the cross validator. Given the amount of data and resources, we used 5 folds as that is statistically proven to be a good standard. We did not pass a pipeline to a cross validator but instead passed a cross validator to our pipeline. This improved the run time tremendously. Otherwise, the cross validator will build each item in the pipeline model for every model it is validating.

SVM:

Training Model AUCROC from Cross-validation: [0.873603528431818, 0.8694270749988696, 0.8716462675738617, 0.9060798749044312, 0.9066123900624242, 0.908365690917429, 0.9122503681166744, 0.9135501968054897, 0.9153293741603081]

Gradient Boosting:

Training Model AUCROC from Cross-validation: [0.9057439257994677, 0.9066722683491294, 0.9074016172577657, 0.908412180727588]

Random Forest:

Training Model AUCROC from Cross-validation: [0.9126279740025589, 0.9135875263160186]

Optimizing PySpark

- **Enough Memory:** We started by making sure the driver and executors have enough memory and cores for processing the data
- **Caching:** To reduce computational resources, as a rule of thumb, we cached our data whenever stored in dataframes and will be reused. Especially while performing *groupby* or *count* functions. Otherwise, spark rebuilds the data and performs the function. When cached, the data will be grabbed from memory and that stage will be skipped.
- **Parallelism:** While performing cross validation to optimize the performance we set training to be done in parallel by using parallelism. This significantly improved the cross-validation time.
- **Efficient hyperparameter tuning:** Though we used parallelism, we had to be efficient about hyperparameter tuning. We had to streamline the number of parameters we wanted to use in grid search. Especially while using trees, some of the parameters can take a lot of computational resources slowing down the performance. Therefore, we researched on the parameters which are the most important and those which can be set without hyperparameter tuning. One of them was selecting optimal principal components. We experimented starting with 10 components and we saw similar results opposed to 30 which was being selected in hyperparameter tuning. We have only 106 features and 10 components seemed reasonable because most of the variance was explained by the first component. We demonstrated how to find the optimal k for the PCA in the SVM model, but we decided not to do the same for the trees.
- **Crossvalidator in the pipeline:** We also identified that when a pipeline model is passed into a cross validator, it builds the pipeline for each model it is validating. Therefore, we changed the approach to perform cross validation on the model for required hyperparameters only. We passed the cross validator as the final stage in the pipeline. This skipped the stage of rebuilding the data for each model and saved much time, especially for the trees.

We used the spark user interface to understand the configuration of jobs and the stages. This was very helpful in understanding where the optimization is required. It also helps in understanding if the jobs are still running or if there was any issue with the job. An example of our observation after tuning the

crossvalidator is steps like below which were skipped instead of being performed for every fold validation.

Figure 9: Spark UI showing the skipped stages while the job is running

Model Building

We followed a consistent procedure as listed below to build our models:

1. We instantiated our models with the label column and input columns assembled using *Vector Assembler*.
2. To make the models comparable, we chose to use the principal components as our input feature vector for all models, after the benchmark model.
3. We created a pipeline with stages which contained the scaled data, PCA model, and the classification model.
4. We used *ParamGridBuilder* to add the various hyperparameters.
5. We used *Binaryclassificationevaluator* to evaluate the model predictions and generate area under the curve. We manually calculated other metrics.
6. We used *CrossValidator* to perform 5-fold cross validation with all the above.

Evaluation

We used the cached test data to make predictions on each model and evaluated the model using the classification metrics F1, Precision, Recall, Accuracy and AUC as shown in F*igure 10* below. We believed that predicting educated or not educated is equally important, so the threshold was set to be default at 0.5 to optimize the accuracy and minimize the overall error rate for all models.

Results

Model Performance

The final results of each of the four models are summarized below.

Model	Parameters	Threshold	AUROC	Accuracy	Recall (TPR)	Precision	F-1
Log Reg	regParam=0.1, elasticParam=0.8	0.5	0.685	71.7%	73.0%	71.4%	72.2%
SVM	regParam=1	0.5	0.847	79.7%	65.2%	92.0%	76.3%
Gradient Boosting	maxIter=15, maxBins=20, maxDepth=10	0.5	0.784	79.6%	75.7%	82.3%	78.9%
Random Forest	numTrees=30, maxDepth=15	0.5	0.790	80.2%	76.4%	82.8%	79.5%

Figure 10: Model Evaluation Metrics

Conclusions

Best Model

Based on our goal of identifying people with high school or less education, and providing them resources, we chose our best model based on the highest recall rate (TPR). With a higher recall rate, we would minimize the case of False Negatives, which is predicting someone isn't educated, when they actually are. We believe it would be more costly to provide resources in areas that don't need them (more educated) than areas that do. Therefore, selecting the Random Forest model with the highest accuracy, highest recall, and second highest precision among our models is recommended. The Random Forest model is also the most balanced among the models for predicting both classes and seemed to perform well with minimal hyperparameter tuning. If we had more resources, the Random Forest model would most likely produce even better results.

Indicators Beyond Geography

Our initial hypothesis considered the region (PUMA) to be the best predictor of education level, but PCA identified many fields had on education (number of hours worked, number of mothers in household, occupation HWSEI - Hauser and Warren Socioeconomic Index (SEI), place of work type). This concludes that the prediction of education is more complex than geographical boundaries.

PySpark Benefits

The consolidated code took 2.5 hours to process the sample of data. Some benefits of using PySpark include:

- Enabling to process large amounts of data in a distributed manner which is not easily achievable otherwise.
- Pipelines/functions made it very convenient to test 4 different models with many hyperparameters
- Cross validation was quick through use of threading/multiple cores

Future Considerations

Future considerations for this project given more time and computational resources are:

- Use all the south region data, or a different sample of the data to see if results are consistent.
- Experiment with tuning shuffle partitions in attempt to make our queries run faster.
- It is a good practice to serialize the data when the data is large. Therefore, we could consider saving the data in parquet format instead of a csv.
- Programmatic changes that we suggest applying are to use *OneHotEncoder* on the other independent variables to transform them from continuous to binary.
- This analysis was on the South Region of the US. A future consideration is to apply the code to data from other regions.
- In addition, applying multi-class prediction to our dependent variable, and predict the education level itself, versus the binary indicator we chose (educated past 12 grade or not) may help pinpoint where education is stopping.

Sources

https://tsmatz.github.io/azure-databricks-exercise/exercise04-hyperparams-tuning.html
https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.tuning.CrossValidator.html
https://towardsdatascience.com/6-sklearn-mistakes-that-silently-tell-you-are-a-rookie-84fa55f2b9dd
https://towardsdatascience.com/all-about-feature-scaling-bcc0ad75cb35
https://towardsdatascience.com/5-ways-to-add-a-new-column-in-a-pyspark-dataframe-4e75c2fd8c08
https://spark.apache.org/docs/2.3.0/api/python/pyspark.ml.html#pyspark.ml.tuning.CrossValidator
https://runawayhorse001.github.io/LearningApacheSpark/classification.html#random-forest-classification
https://runawayhorse001.github.io/LearningApacheSpark/classification.html
https://shihaojran.com/distributed-machine-learning-using-pyspark/
https://www.analyticsvidhya.com/blog/2020/06/auc-roc-curve-machine-learning/
https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.sampleBy.html
https://towardsdatascience.com/exploratory-data-analysis-eda-with-pyspark-on-databricks-e8d6529626b1
https://www.kaggle.com/tientd95/advanced-pyspark-for-exploratory-data-analysis
https://stackoverflow.com/questions/22984335/recovering-features-names-of-explained-variance-ratio-in-pca-with-sklearn
https://www.py4u.net/discuss/218858
https://datascience-enthusiast.com/Python/PCA_Spark_Python_R.html
https://tsmatz.github.io/azure-databricks-exercise/exercise04-hyperparams-tuning.html
https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.tuning.CrossValidator.html
https://towardsdatascience.com/pre-process-data-with-pipeline-to-prevent-data-leakage-during-cross-validation-e3442cca7fdc
https://awesomeopensource.com/project/adornes/spark_python_ml_examples
https://spark.apache.org/docs/latest/ml-tuning.html
https://sparkbyexamples.com/pyspark/pyspark-rename-dataframe-column/
https://spark.apache.org/docs/latest/ml-classification-regression.html#random-forest-classifier
https://people.stat.sc.edu/haigang/sparkCaseStudy.html
https://sparkbyexamples.com/pyspark/pyspark-cast-column-type/
https://george-jen.gitbook.io/data-science-and-apache-spark/binarizer
https://towardsdatascience.com/effortless-hyperparameters-tuning-with-apache-spark-20ff93019ef2
https://kb.databricks.com/machine-learning/speed-up-cross-validation.html
https://www.ateam-oracle.com/multiclass-text-classification-crossvalidation-with-pyspark-pipelines
https://intellipaat.com/community/6592/what-does-stage-skipped-mean-in-apache-spark-web-ui
https://towardsdatascience.com/predict-customer-churn-with-pyspark-and-machine-learning-981d1eedb00b
https://dhiraj-p-rai.medium.com/logistic-regression-in-spark-ml-8a95b5f5434c
https://towardsdatascience.com/predict-customer-churn-with-pyspark-and-machine-learning-981d1eedb00b
https://towardsdatascience.com/apache-spark-caching-603154173c48