ETL Report

Introduction

The purpose of this project is to test whether machine learning would be a viable alternative (or complement) to traditional life insurance underwriting. To that end two data sets were collected: one from the census detailing life insurance costs (to companies) and payouts to better understand the expenses related to the life insurance industry. Second was a data set from Prudential Financial which features 118 variables of anonymized data from over 60,000 applicants. The latter data set was used to model various machine learning algorithms to find the one with the most accuracy and robustness.

The data sets were both cleaned and their respective data was loaded into SQL for further analysis.

Data Sources (Citations)

1. *Administrative Expenses and Benefits Paid for Life, Health, and Medical Insurance Carriers for the U.S. in 2017*. Explore census data. (n.d.). Retrieved October 26, 2021, from https://data.census.gov/cedsci/table?q=ECNADMBEN2017.EC1752ADMBEN&tid=ECNADMBEN2017.EC1752ADMBEN&hidePreview=true.
2. *Prudential Life Insurance. (2015, November). Prudential Life Insurance Assessment Version 1*. Retrieved October 26th, 2021 from: https://www.kaggle.com/c/prudential-life-insurance-assessment/overview/description

Extraction

Census Dataset:

We were given various links to different census datasets to make use of in our capstone. We chose to use the link titled “Economic Data from the Census”. From there we navigated to the “Finance and Insurance (NAICS Sector 52)” page, and then finally to the link titled “Finance and Insurance: Administrative Expenses and Benefits Paid for Life, Health, and Medical Insurance Carriers for the U.S.: 2017” located under “Miscellaneous Statistics”.

Once we got to the actual data table, we downloaded the dataset as a CSV to our local pc. From there, we uploaded our CSV as a blob into the group3-capstone container within the gen10dbcdatalake. We then loaded our census CSV into a python dataframe within a datatabrick in order to transform the data.

Prudential Life Insurance Dataset:

This dataset was found on kaggle.com as part of a competition to see how accurately machine learning was in predicting the risk category placement (ordinally encoded variable from 1-8) given several other parameters of life insurance applicants. There are 59,381 such applicants (or records) and 126 columns (features). All of these features are summarized in the figure below:

Variable	Description
Id	A unique identifier associated with an application.
Product_Info_1-7	A set of normalized variables relating to the product applied for
Ins_Age	Normalized age of applicant
Ht	Normalized height of applicant
Wt	Normalized weight of applicant
BMI	Normalized BMI of applicant
Employment_Info_1-6	A set of normalized variables relating to the employment history of the applicant.
InsuredInfo_1-6	A set of normalized variables providing information about the applicant.
Insurance_History_1-9	A set of normalized variables relating to the insurance history of the applicant.
Family_Hist_1-5	A set of normalized variables relating to the family history of the applicant.
Medical_History_1-41	A set of normalized variables relating to the medical history of the applicant.
Medical_Keyword_1-48	A set of dummy variables relating to the presence of/absence of a medical keyword being associated with the application.
Response	This is the target variable, an ordinal variable relating to the final decision associated with an application

Figure 1 Table displaying the feature (variable) name and description as provided by Prudential. The table was taken from: https://towardsdatascience.com/life-insurance-risk-prediction-using-machine-learning-algorithms-part-i-data-pre-processing-and-6ca17509c1e

The dataset was obtained as a CSV and uploaded to the gen10dbcdatalake in the same container and in the same way as the census data above. Transformation of the data was done in databricks.

Transformation

Note:

1. All Transformation steps were done in databricks using python 3.x
2. Below are all of the necessary import statements:
   a.
```
Cmd 1
Import Statements

1  import pandas as pd
2  from pyspark.sql import functions as F
3  import os.path
4  import numpy as np
5  from sklearn.impute import KNNImputer
6  from pyspark.sql.functions import col
7  from pyspark.sql.functions import when
```
3. A Mount point was set up to access the data lake where both data files were as according to the screenshot below:
   a.
```
###### Mount Point 1 through Oauth security.https://adb-593121260067740.0.azuredatabricks.net/?o=593121260067740#
storageAccount = "gen10dbcdatalake"
storageContainer = "group3-capstone"
clientSecret = "~bJ7Q~KslVT~sAmHkOLXL0oeTp1ZkAcndtHPr"
clientid = "2ca50102-5717-4373-b796-39d06568588d"
mount_point = "/mnt/group3"


configs = {"fs.azure.account.auth.type": "OAuth",
       "fs.azure.account.oauth.provider.type": "org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",
       "fs.azure.account.oauth2.client.id": clientid,
       "fs.azure.account.oauth2.client.secret": clientSecret,
       "fs.azure.account.oauth2.client.endpoint": "https://login.microsoftonline.com/d46b54b2-a652-420b-aa5a-2ef7f8fc706e/oauth2/token",
       "fs.azure.createRemoteFileSystemDuringInitialization": "true"}
```
4. For a more detailed view of how the steps were done (i.e., the exact code used) please refer to the databricks notebook entitled “Data Processing” on the projects GitHub Repo.

<u>Census Dataset:</u>

1. Setup a mount point in a data brick to connect to the group3-capstone container.
2. Load the census CSV to the databrick from the gen10dbcdatalake into a spark dataframe.

3. Using the spark .drop() function, drop the columns “Geographic Area Name (NAME)”, “2017 NAICS code (NAICS2017)”, “Response coverage of administrative expenses inquiry (%) (EXPADMIN_COV)”, and “Response coverage of benefits paid inquiry (%) (EXPBN_COV)”.
4. Using the spark .filter() function, filter the dataframe to only contain the rows that have a value of “Direct life insurance carriers” within the “Meaning of NAICS code (NAICS2017_LABEL)” column.
5. Change the “X” values in the columns “Number of firms (FIRM)”, “Sales, value of shipments, or revenue ($1,000) (RCPTOT)”, and “Benefits paid ($1,000) (EXPBENP)” to None values. This can be done by utilizing the spark when() function. Here is an example of changing the values using the “Number of firms (FIRM) column: df = df.withColumn("Number of firms (FIRM)",when(col("Number of firms (FIRM)").isin('X'),None).otherwise(col("Number of firms (FIRM)")))
6. Write the transformed spark dataframe back to the group3-capstone container as a CSV.
7. After loading the data into Power BI, use Power Query to do a find and replace for each value in the “Meaning of Type of administrative expenses and benefits paid code (EXPADMBNTYPE_LABEL)” column so that each value is shorter. This will make the visualizations more readable.
8. While still in Power Query, for the columns “Number of firms (FIRM)”, “Sales, value of shipments, or revenue ($1,000) (RCPTOT)”, “Administrative expenses ($1,000) (EXPADMIN)”, and “Benefits paid ($1,000) (EXPBENP)” change the data type to whole number.

<u>Prudential Life Insurance Dataset:</u>

1. Using the established mount point above, the data was accessed using:
    a. df = spark.read.csv(mount_point+ "/Prudential Life Insurance.csv", header = "true")
    b. And was stored as a pyspark data-frame
2. Using the following code:
    a. df_agg = df.agg(*[F.count(F.when(F.isnull(c), c)).alias(c) for c in df.columns])
    b. display(df_agg)
    c. A table was made displaying the null counts in each column.
    d. The columns with more than 15% (~8900) values missing were dropped, namely:
        i. Employment_info_6, Insurance_History_5, Family_Hist_2-5, Medcial_History_10,15,24,32
3. The aforementioned columns were dropped using the .drop() method (see data brick or pyspark documentation to see how this would be done).
4. Next any duplicate rows were dropped using the .dropDuplicates() method.
5. Next the data-frame was converted to a pandas data-frame as it is small enough to not have to worry about speed and since the next few methods for cleaning will require pandas (as far as I know). To convert to a pandas data-frame we simply use:
    a. dfP = df.toPandas()
6. Next any records with more than 50% of the values missing are removed using:
    a. dfP.dropna(thresh = 59, inplace = True)

7. Lastly there were still missing values, all of which were categorical, and so they were imputed using a KNN imputer (again see data brick for exact code).
8. The imputer returns a numpy array so that was turned back into a pandas dataframe and then a spark one (again see data brick for how).
9. Lastly the cleaned data was returned back to the data lake as follows:
    a. final.write.csv(mount_point + '/CleanedInsuranceData', header = "true")

Load

Once our datasets were cleaned, we loaded both of them into their own SQL table. We then used each SQL table with Power BI in order to create various visualizations of our data. The load process for each dataset is described below.

Note: The load process to SQL was done using databricks and data factory as shown in the figure below:

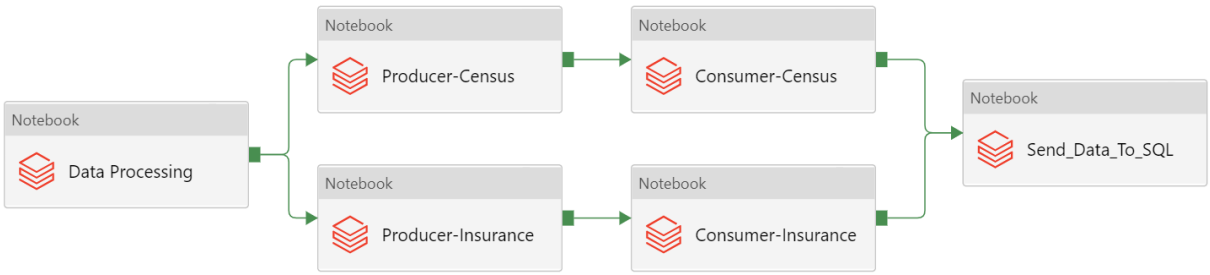

Figure 2 Using the databricks created from before the process of cleaning and sending the data to SQL was linked via several databrick notebooks.

Census Dataset and Prudential Life Insurance Dataset:

1. The Data Processing notebook is described in the Transformation section above.
2. Once data processing is complete the next step carries out both producer steps in parallel for both data sets.
3. The producer sends messages to the consumer which consumes the data, performs any last-minute cleaning if necessary, and then sends the data to SQL.

Note: For the exact code used to send, receive, upload the data into SQL please read the producer and consumer data brick notebooks for each data set in the code folder of the projects GitHub repo.

Also, no ERD was created for these data sets as they don't relate to each other. In addition, while each data set can be normalized it was deemed unnecessary to do so as the census data set was quite small, and the prudential data set was heavily encoded (anonymized), as well as only required for the machine learning portion of the project. The prudential dataset is also static and does not have any more information to add in so normalization would not be a worthwhile exercise here.

Conclusion

The purpose of this project is to determine whether using machine learning can speed up the process of insurance underwriting to save costs and remove any subjectivity that may be involved in the process. To that end two data sets were collected: Census data detailing administrative expenses and life insurance payouts and data from Prudential Financial which gives anonymized data of over 60,000 life insurance applicants and what risk score they placed into. These data sets were cleaned and loaded into SQL whereby they could be further used for machine learning and creation of visuals via power BI.