Auto Insurance Claim Prediction for Porto Seguro

TABLE OF CONTENTS

1. Discover

1.1 Understanding the problem

1.2 Why ML over other solutions

1.3 KPI and metric(s) to track

1.4 Data Collection

1. Play

2.1 Git Repo

2.2 Create Package

2.3 EDA

2.4 POC/Model Selection

2.5 Decide on Model

1. Develop
2. Deploy
3. **Discover**

**1.1 Understanding the problem**

Porto Seguro wants to improve customer satisfaction by tailoring auto insurance prices for drivers. We need to know which customers are likely to file a claim(bad drivers) and which are likely to not(good drivers). The final goal is to increase insurance prices for bad drivers and reduce the prices for good drivers.

This project aims to use machine learning to predict which customers are likely to file a claim.

**1.2 Why ML over other solutions**

* Adaptability
* Scalability
* Grasp Complex Patterns

**1.3 KPI and metric(s) to track**

1. KPI**:** Increase customer satisfaction by 60% over the next year.
2. Metric:

* F1: balance between reducing False Positives and False negatives as both are costly) and robust to class imbalance
* AUC: Robust to Class Imbalance

**1.4 Data Collection**

The data was collected from [Kaggle](https://www.kaggle.com/competitions/porto-seguro-safe-driver-prediction/data) and uploaded to AWS S3

**1.4.1 About Dataset**

In the train and test data, features that belong to similar groupings are tagged as such in the feature names (e.g., ind, reg, car, calc). In addition, feature names include the postfix bin to indicate binary features and cat to indicate categorical features. Features without these designations are either continuous or ordinal. Values of -1 indicate that the feature was missing from the observation. The target columns signifies whether or not a claim was filed for that policyholder.

**1.4.1.1 File descriptions**

* Train.csv: contains the training data, where each row corresponds to a policyholder, and the target column signifies that a claim was filed.
* Test.csv: contains the test data.

**2. Play (1\_EDA.ipynb)**

**2.1 Git Repo**

* Create Repo in GitHub and open the project folder on local machine
* Run git init in the terminal
* Create master branch in github and set it to default(only if the default branch in local and GitHub are different)
* Delete main branch in GitHub
* Run commands in Terminal: git remote add origin <git url>, git pull origin master
* Create dev branch: git branch dev => git checkout dev

**2.2 Create Package**

A package will be created and installed into jupyter and vscode for use. It contains reusable functions for tasks like visualization, preprocessing, deployment, etc.

Process:

* Create the folder (custom\_package)
* Create a sub-folder(package)
* Create a setup.py file within custom\_package and input values
* Create modules and functions for the package
* Run these: cd package, pip install build, python -m build, cd dist, pip install <package-file.whl> –force-reinstall, cd ../..

**2.3 EDA**

**2.3.1 Connect to AWS S3**

1. Create IAM User in AWS:
2. Open IAM service
3. Click on users in the left panel > Create user
4. Input user name(e.g. jupyter\_access)
5. Select policy (e.g. administrator access) > Create user
6. Select newly created user and click on create access key
7. Select use case
8. Download .csv file
9. Connect to S3 in jupyter:
10. Create s3 client object:

s3\_client = boto3.client('s3',aws\_access\_key\_id=secret['Access key ID'],aws\_secret\_access\_key=secret['Secret access key'])

1. Input name of bucket and file:

bucket = 'auto-insurance-data-x'

file\_name = "train.csv"

1. Get object/file from s3:

s3\_clientobj = s3\_client.get\_object(Bucket=bucket, Key=file\_name)

3. Create data\_retrieval module in custom package:

1. Create data\_retrieval.py module under data\_retrieval sub-package
2. Within data\_retrieval.py, create the function to retrieve data from an s3 bucket
3. Save and update the setup.py file to include the new sub-package

from setuptools import setup

setup(

name = 'package',

version = '0.1',

description = 'Useful functions for training and deployment',

author\_email = 'oamenmodupe@gmail.com',

packages = ['package.feature\_engineering'],

install\_requires = ['numpy', 'pandas', 'scikit-learn', 'matplotlib', 'mlflow']

)

1. Build the package: cd custom\_package> python -m build > cd dist > pip install pip install package-0.1-py3-none-any.whl
2. Import new function in jupyter notebook for use

2.3.2 Replace -1 with np.nan

The dataset came with null values encoded as -1. These values will be replaced with np.nan so they can be handled properly.

2.3.3 Stratified Sampling

In stratified sampling, researchers divide subjects into subgroups called strata based on characteristics that they share. We start with splitting to avoid data snooping bias.

Here, we stratify the target variable and split it into four datasets:

1. Full Train: Full training dataset (reduced train + validation set). This set will be used to train the chosen dataset.
2. Reduced Train: This dataset will be used for EDA and to experiment and try out different algorithms and feature engineering techniques before deciding on the best-performing one.
3. Test: This will be used to test the performance of the main model
4. Validation: This will be used to evaluate models and for hyperparameter tuning

All datasets will be written to S3 using the write\_to\_s3 function created within the custom\_package.

2.3.4 Data Exploration

A random sample of 100,000 was drawn from the reduced train set for faster exploration.

2.3.4.1 Duplicates

The dataset contains no duplicates.

2.3.4.2 Whitespaces

If the column names contains whitespaces, the whitespaces need to be removed for easier feature retrieval. If the dataset has empty strings, it will cause the column type to be inferred wrongly by pandas as an object type instead of the actual data type. The column names and dataset contains no whitespaces.

2.3.4.3 Missing Values

The dataset is 2% nulls, with columns containing 0-69% nulls.

2.3.4.4 Split Data Types

We split the dataset in types, categorical and numerical, and save the feature names to the config file. This is useful so we can easily explore the dataset according to the data types.

2.3.4.5 Univariate analysis

Our dataset is mostly discrete data so it’s hard to get around spotting or reducing cardinality but we notice a significant class imbalance in the target. We notice some outliers in the boxplots.

2.3.4.6 Bivariate analysis

Not much to see here as the column names are uninformative. Chi-Square Test of Independence was performed to see the relationship between the target and categorical variables. This test is used to determine if two categorical variables are independent or if they are in fact related to one another. If two categorical variables are independent, then the value of one variable does not change the probability distribution of the other. If two categorical variables are related, then the distribution of one depends on the level the other. This test measures the differences in the observed conditional distribution of one variable across levels of the other, and compares it to the marginal (overall) distribution of that variable.

2.3.4.7 Linear Separability

We use seaborn’s pairplot to plot variables against each other with the hue set to the target to see if a linear decision boundary can we drawn. This is important as it helps in confirming the assumptions of Linear Models. If there is no linearity, we transform the data or use non-linear models.

2.3.4.8 Stats

We do this especially to get an idea of the skew in our data. It also helps in knowing what type of imputation to use. We can use mean imputation in a case where the data is normal and median imputation when it is not. In this case, the data is skewed, so, we will employ median imputation.

2.3.4.9 Correlation

This is using a custom function in our package. We pass in the data, correlation type, figsize, and annot parameters. In this case, we check for linear/pearson and non-linear/spearman correlation.

2.3.4.10 Normality

We use two methods here to check for normality, The Shapiro-Wilk test and Q-Q plot. Shapiro-Wilk’s null hypothesis is that the data is normal. We pass this test for each column and reject or accept the null hypothesis according to the pvalue. The Q-Q plot shows quantile points against the theoretical normal line. If all or most values lie on this line, we can assume the data is normal or close to normal.

2.3.4.11 Outliers

We use two methods here, the IQR Proximity Rule and ZScore. For the IQR method, we consider values above the 75th percentile and 25th percentile, outliers.

For the ZScore, we consider values with a zscore outside the threshold (+3, -3), outliers.

2.3.4.12 Write to Json

We write the config to json

**2.4 Model Selection/POC (2\_model\_selection.ipynb)**

2.4.1

2.6 Decide on Model

Resources

1. <https://stackoverflow.com/questions/58402319/connect-jupyter-notebook-locally-to-aws-s3-without-sagemaker>
2. <https://stackoverflow.com/questions/54855479/s3-object-throws-typeerror-sequence-item-0-expected-str-instance-tuple-foun>
3. <http://localhost:8888/notebooks/OneDrive/Documents/DATA%20PROJECTS/Project_Determining_Risk_tolerance_category.ipynb>
4. <https://sites.utexas.edu/sos/guided/inferential/categorical/chi2/#:~:text=If%20two%20categorical%20variables%20are%20independent%2C%20then%20the%20value%20of,on%20the%20level%20the%20other>.
5. <https://www.linkedin.com/advice/3/how-can-you-remove-outliers-from-dataset-using-wyx2e#:~:text=To%20spot%20outliers%20with%20the,data%20in%20a%20normal%20distribution>.
6. <https://www.geeksforgeeks.org/reading-and-writing-json-to-a-file-in-python/>
7. <https://medium.com/analytics-vidhya/appropriate-ways-to-treat-missing-values-f82f00edd9be>
8. <https://www.learningtree.com/blog/interpret-q-q-plot/>