Capstone Proposal

Machine Learning Engineer Nanodegree
Diego García Lozano
2020/04/15

1. Domain Background

For the final project I have decided to make the Arvato's problem because of the following reasons:

- I wanted to develop a solution for a **real problem**, so the classifier for dogs was discarded. It could be a real business problem but I think is a strange business case
- I preferred to choose one of the projects proposed by Udacity over searching another one, in order to **avoid bias** (for example, choosing an easy one)
- So finally my options were either Arvato or Starbucks. Arvato project has a **Kaggle competition** associated and I thought it would be more self-motivating, so I chose it

2. Problem Statement

Arvato, which is a company that provides **financial solutions**, has a problem that consists on create a **customer segmentation**.
We have available different data from both customers and non-customers (german people), so we need to provide **insights** in order to know what kind of people is more likely to become a customer.
After that, we will develop a **supervised machine learning model** to classify people with different characteristics, and finally we will upload some solutions with test data in order to check our accuracy.

So the problem keys are:

- Is a person with certain characteristics a **potential customer**?
- If two people has exactly same characteristics, will we think that both have the same behaviour in order to avoid potential **regulatory problems**?
- Can we use **Machine Learning** to solve the problem?
- Do we have **data** that provides information in order to check our results?
- How are we going to **measure** that results?

Our potential solution will be a **supervised Machine Learning model for classification**, where our features will be the people characteristics from both german and customers people, and the label will be binary, whether the person is a potential customer or not.

3. Datasets and Inputs

Arvato provides us different datasets to develop a new Machine Learning model. We can divide it in **three main parts**:

- Insights data:
  - *azdias*: data with **german population** characteristics. Over 890000 rows and 366 features
  - *customers*: data with company **customers**. Over 190000 rows and 369 features
- Train data. Data that we'll use to train our model:
  - About 42000 rows and 367 features
- Test data. We'll use this dataset to check our results from the previos model:
  - About 42000 rows and 366 features (the target is not available here)

    - It has about 41500 non-customers and 500 customers. It seems like an **imbalanced** problem. Should we use data from azdias and customers to improve the model?

4. Solution Statement

Our final solution will be a trained **Machine Learning model** that will be able to differentiate between potential and non-potential customers.

The datasets contains **many variables** (almost 400) and NaNs, so we will use different techniques to choose which variables will the model use after **cleaning the data**.
To make sure that the solution is **replicable**, we'll make use of seeds to create pseudo random numbers when initializing algorithms or splitting data.

5. Benchmark Model

In this problem, benchmarking our model is easy due to the fact that exists a Kaggle competition associated to it. So once we have developed the model, we'll update our results to check our accuracy.
It we take a look at **Kaggle leaderboard**, an AUC **between 0.6 and 0.8** seems a reasonable result to build a good enough solution.

6. Evaluation Metrics

We will measure it with **AUC** (Area Under Curve), because is the metric that is used in the Kaggle competition.
It seems reasonable because it's likely to be a **ranking problem**. The company needs to know which people is more likely (has more probabilities) to become a customer in order to make actions, for example a marketing campaign.

7. Project Design

We can summarize our workflow in several steps, that we'll include in a Python notebook:

- Reading *insights data* (*azdias* and *customers*). Take a look at some characteristics and ask ourselves some questions:

    - Have the data **null fields**?
    - Do the null data follow a **pattern**?
    - Is the filled data **cleaned** and correct?
    - Do we have the **same features** for both datasets? We should use same characteristics for both of them
    - How do we deal with **categorical features**?
    - Can some of the **categorical features** be already encoded? Should we still use this encoding or change it?
    - Do we have **datetime features**? Should we extract new features?
    - Can we create new **interaction variable** from others?
    - Both *azdias* and *customers* features follow **same distributions**?
    - Do they have same **number of nulls**?

  To approach this, we'll use some techniques and Python libraries:

    - Pandas to clean the data: fill or remove NaNs, manage datetime features, create new variables…
    - Scikit-Learn to make use of some algorithms: KMeans, PCA and other unsupervised techniques, categorical encoders…
    - Numpy, Scipy and itertools for other functionalities

- Training data:

  - Do we have **enough data**?
  - Does it follows the **same patterns and distributions** from *azdias* and *customers*?
  - Should we make use of *azdias* and *customers* data to train our model?
  - Do we need to deal with **imbalanced data** or, just because we are going to predict probabilities,
    we don't need to take care of it?
  - We should train our model and test its capabilities with a **cross validation** method. We don't have many positives cases,
    so the folds should be **stratified** in order to follow same distribution
  - We'll try typical algorithms in Kaggle competitions, like **boosting trees**. CatBoost could be a good choice in order to avoid take care of categorical features

  To approach this, we'll use some techniques and Python libraries:

  - Pandas to clean the data: fill or remove NaNs, manage datetime features, create new variables…
  - CatBoost to implement a boosting tree classifier that also encodes categorical features for us
  - Numpy, Scipy and itertools for other functionalities

- Test data:

  - Does test data follow **same distribution** as train?
  - If we upload our results to **Kaggle**, do we obtain a similar result as cross validation in train data? If this is the case, test data, and specially, **private** test data will follow the same distribution, so it will be easier for us to check our results