Introduction

To analyze and understand the important features for successful startups, I trained XGBoost Regression model which predicts whether the company will succeed or fail.

Why XGBoost?
It can automatically provide estimates of feature importance from a trained predictive model.

Data Preprocessing

Some problems of the dataset and important points in data preprocessing.

- Data contains a lot of NaN values, but also it contains other values that should be considered as NaN, such as "No Info", "Not Applicable", "Nothing", "None" and , "unknown amount". This is solved during reading the csv file by setting `values_na` in `pd.read_csv()` function

- More preprocessing is needed for 'object' type columns of the dataset. Which can be divided into the following general types

    - **Type1 -** each row of the column contains multiple features from the features set of the column: Features in the rows are separated by '|'-s

      For extracting features set from the particular column `get_uniques()` function is written.

    - **Type2 -** each row of the column contains exactly one feature from the features set.

  These two types of the column can be checked by comparing unique values of the column with uniques returned from `get_uniques()` function. We are dealing with Type2 columns If mentioned arrays are the same for that column.

  For Type2 columns `OneHotEncoder` could be applied, but columns with *"both"* value exist, consequently another function is needed for creating new features columns based on the main column.
  `split_column()` function does this for both *Type1* and *Type2* columns.

- In some columns *"No"* value is equivalent to NaN, but also exists columns where *"No"* is a useful feature(usually *"Yes" & "No"*). So, *"No"* values must be removed from the list of uniques of the first columns. Besides *"No", "both"* also have to be removed for all columns. `filter_features()` function does these.

- A lot of missing values in some columns. So, if the quantity of NaN-s in the column is more than half of the length of the dataset, the column is dropped.

- Almost all rows are different. These could be solved by applying word2vec to the features and grouping them, or by just dropping that column. In this solution column is dropped if the unique element count is more than ⅔ of the length of the dataset.

- The column is dropped if one of the value_counts is more than length of the dataset. For this `drop_columns()` function is defined.

- All NaN-s could be filled either with column `means` or using `forwardfill` method depending on the datatype of the column. I filled only numeric values with the mean.

After data preprocessing dataset containes **170** features and **472** rows

Training the model

XGBRegressor with gradient boosted trees was trained on 80% of data.

The model was tested on 20% of data. Model has **82.1%** accuracy.

As was mentioned in the introduction the main purpose of using XGBoost is that after the boosted trees are constructed, it is relatively straightforward to retrieve importance scores for each attribute.

From *Fig. 1* can be seen that top 5 driving features are

1. Internet activity score
2. Number of recognitions of founders and co-founders
3. Time to first investment
4. Percent skill business strategy
5. Last funding amount

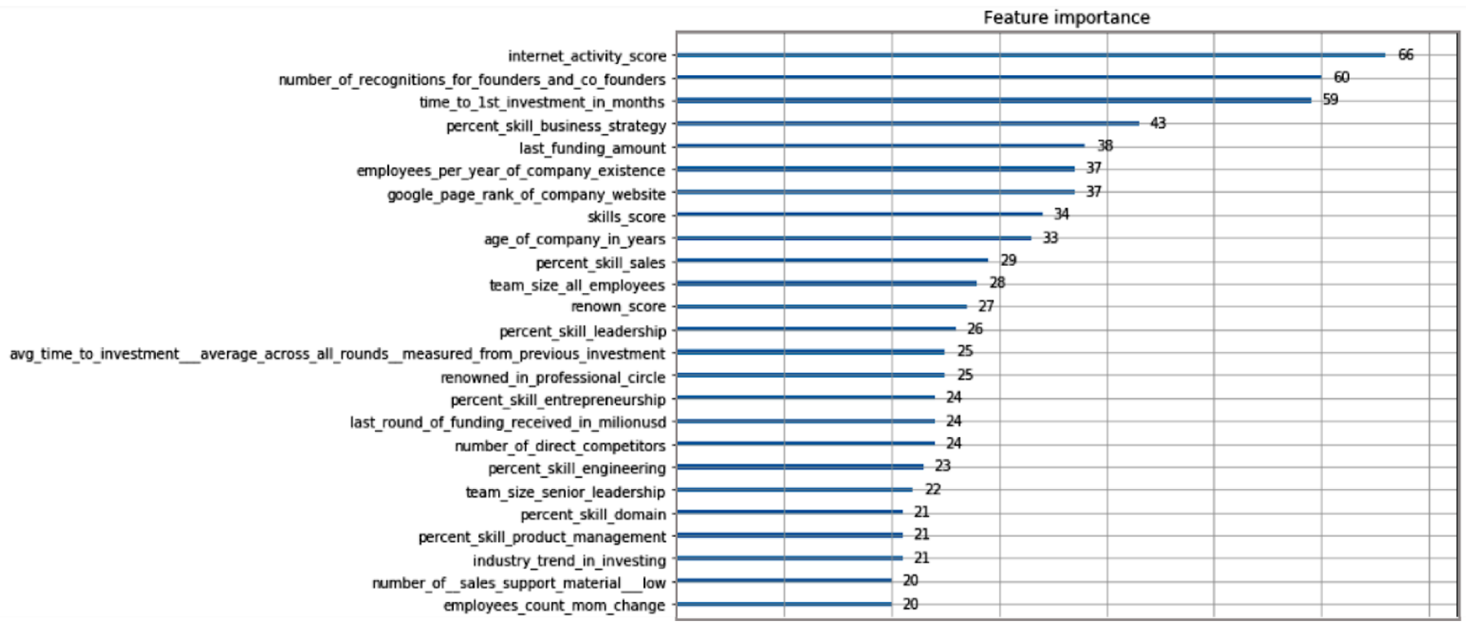

Fig. 1 Plot of the feature importances (the file could also be found in the directory)