DATA SCIENTIST NANODEGREE

CAPSTONE PROJECT

PREDICTION OF COMPANY FAILURES IN FRANCE USING PUBLICY AVAILABLE INFORMATION IN COMMERCIAL REGISTERS

by Aurélien Fauvel

Executive Summary:

The objective of this project is to be able to predict which French companies will go out of business within 12 to 24 months, using publicly available information in the commercial registries.

We will see that using a Random Classifier we are able to build a very interesting / performing model for any given year, improving the precision of by several factors; however this model does not generalize well to previous or later years. In essence this project didn't reach the desired outcomes and the built model does not bring an advantage that can be used in real life.

The process of building up and evaluating this model was, despite the final outcome, very fruitful and provided me with some very interesting insights and ideas to improve further on this project.

---

1. PROJECT DEFINITION

1.1 Project overview

Background:
The restructuring "industry", the business of buying companies in relatively bad shape and turning them around, has been flourishing in Western Europe for some years. I have been active in this field for close to 10 years now and through the many restructuring situations I experienced was able to learn a lot.

Buying such companies is however quite risky and tricky, as the success rate can be low and the stakes (financial and human) quite high. Some private equity funds have specialized in that domain with essentially two expertise:

- Deal sourcing: how to find the right company to buy
- Operational improvement: how to practically turn a company around when bought

Getting access to and finding the "right" companies to buy is essential for the success of any private equity fund. One key criterion is to find a company that is in not to desperate condition so that a turnaround is possible and ideally one company that no other fund has on its radar.

Objective:
In this project my objective is / was to find a (data science) way to increase the odds of finding those companies before they are put to sale and they reach a bad financial condition.

Given my background (where I come from and where I live), I have been looking for information either in France or in Germany to answer my question. As I have found the relevant information in France, I have decided to focus myself on the French "case".

Data:
For this project I will use the French commercial registry (public) information, available under: https://opendata.datainfogreffe.fr/explore/?sort=modified

Basically I will work with two different sources of information:
- Information provided by companies when registering / publishing their annual results (one file per year for all French companies)
- Information about the list of companies which went out of business in a given year

Licencing / Use of the data:
Information can be found, in French, under (https://www.etalab.gouv.fr/wp-content/uploads/2017/04/ETALAB-Licence-Ouverte-v2.0.pdf).

Basically the use of those information is free, one only needs to mention where the data has been used from.

1.2 Problem statement

My objective in this project can be summarized as follows:

" Develop a model that enables to significantly increase the odds of identifying French companies that are going to go out of business within 12 to 24 month".

Particularly I will be using the 2017 published financial information of French companies and the list of companies that went out of business in 2019 (January to end of October).

1.3 Metrics

We will see in the Part 1 of the analysis that (fortunately) very few companies go each year out of business (a few % of the ones which can be found on public registries). This means that looking for accuracy will not be of much help, and in this case we are rather interested in the precision of our predictions: e.g. that we correctly predict that one company will go out business in 12 to 24 months (the default is that companies do not go out of business).

Focusing on precision is particularly important in this case because the process of contacting current company owners to potentially buy their company is extremely time intensive and costly. By properly identifying a company in difficulty and likely to go out of business, the odds of making a (restructuring) deal would be higher.

The metric to be optimized will be the prediction precision of companies going out of business in 12 to 24 months. Especially I will be looking at predicting those based on information published in 2017 in the official French registries and the companies that went out of business in 2019 (January to October).

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$

With:
- True Positives: a company correctly predicted going out of business
- False Positives: a company falsely predicted as going out of business

1.4 Structure of the project and files

All the steps of this project are split into 6 notebooks, for structure reasons (so that I don't need to repeat the same steps each time I worked on the project) and memory reasons (many times my laptop blocked for memory reasons when working with ML models).

- CapstoneProject_Part_1: reading files, exploring, cleaning and saving cleaned date into new file.
- CapstoneProject_Part_2: selecting a classifier type for our project
- CapstoneProject_Part_3: optimizing model, testing on test set and 2016 data
- CapstoneProject_Part_3-2: building model on 2016 data, testing it on test set and 2017 data
- CapstoneProject_Part_3-3:explore and test alternatives to find a modelt hat generalized better.

2. DATA EXPLORATION

2.1 Financial information

Size:
The data contains more than 1 Million records, each corresponding to one company that has published its results in 2017. For each record we have 41 columns / characteristics.

```
# Size of dataframe (nb of records and nb of features)
data_2017.shape
```

```
(1154008, 41)
```

Columns / features:
Column names (in French):

```
# Name of columns and feature names
data_2017.columns
```

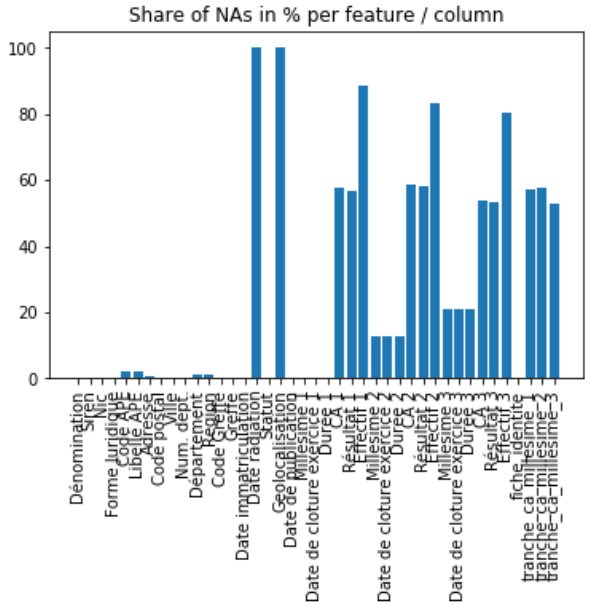
```
Index(['Dénomination', 'Siren', 'Nic', 'Forme Juridique', 'Code APE',
       'Libellé APE', 'Adresse', 'Code postal', 'Ville', 'Num. dept.',
       'Département', 'Région', 'Code Greffe', 'Greffe',
       'Date immatriculation', 'Date radiation', 'Statut', 'Geolocalisation',
       'Date de publication', 'Millesime 1', 'Date de cloture exercice 1',
       'Durée 1', 'CA 1', 'Résultat 1', 'Effectif 1', 'Millesime 2',
       'Date de cloture exercice 2', 'Durée 2', 'CA 2', 'Résultat 2',
       'Effectif 2', 'Millesime 3', 'Date de cloture exercice 3', 'Durée 3',
       'CA 3', 'Résultat 3', 'Effectif 3', 'fiche_identite',
       'tranche_ca_millesime_1', 'tranche_ca_millesime_2',
       'tranche_ca_millesime_3'],
      dtype='object')
```

Columns 1 to 18 are basic information about the company (Company name, ID, legal form, activity, address, and so on). Starting from column 19 (*Date de publication*) are the specific published information (for the last 3 years).

"Siren" is the official company code, and will be our key to identify and merge the other file into this one in the pre-processing part.

NaNs:
One characteristic of this data is the high number of NaNs. Some features have almost only NAs and some (potentially) very important ones contain a lot of NaNs.

As one can see below 14 features have about 50% and more NaNs and another 6 about 20%.




```
# How many NAs are there in the dataset across the features / columns
nas = np.round(data_2017.isna().sum() / data_2017.shape[0] * 100,1)
nas = nas.sort_values(ascending = False)
print(nas)
```

```
Geolocalisation              100.0
Date radiation               100.0
Effectif 1                    88.7
Effectif 2                    83.3
Effectif 3                    80.5
CA 2                          58.6
Résultat 2                    58.1
CA 1                          57.8
tranche_ca_millesime_2        57.8
tranche_ca_millesime_1        57.1
Résultat 1                    56.7
CA 3                          53.6
Résultat 3                    53.3
tranche_ca_millesime_3        52.9
Millesime 3                   21.1
Durée 3                       21.1
Date de cloture exercice 3    21.1
Millesime 2                   12.9
Date de cloture exercice 2    12.9
Durée 2                       12.9
Code APE                       2.1
```

Data types:
Many data have not the correct data type (dates are coded as objects and not dates, strings as object, …) and will need to be processed.

Possible values of important categorical features:
From the many categorical features this dataset contains, I am particularly interested in using 3 of those for this model, as I hypothesize that they might help predict company failures:

- "Num. dept.": this is the department number where the company is registered, e.g. an indication of the location. I believe this can be of interest as some geographical areas in France are economically more difficult and therefore could help predict company failures
- "Code APE": this code represent the activity sector in which the company is active (industry, services, …)

- "Forme juridique": is the legal form of the company. It gives an indication of the capital structure, governance and size of the company.

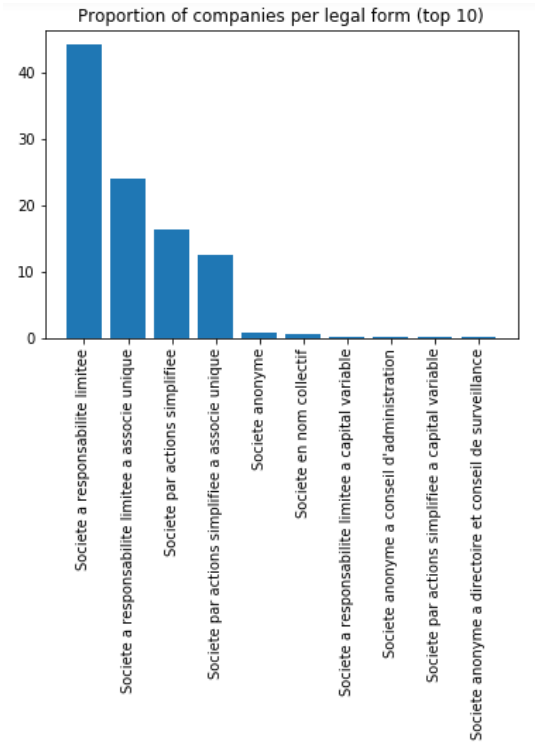
Departments: are well known, a little fewer than 100 for France.

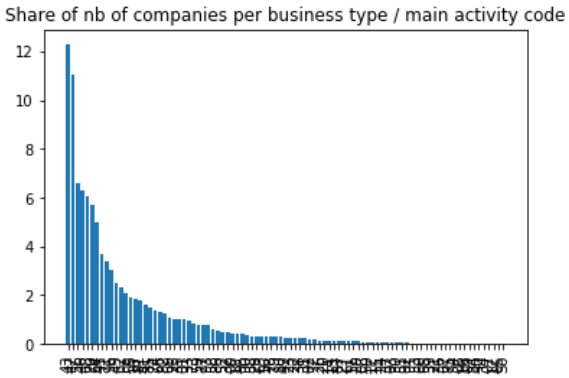
Code APE and Forme Juridique:

```
# Nb of possible values for different features
print("Nb of possible legal forms: ", data_2017["Forme Juridique"].nunique())
print("Nb of possible business types: ", data_2017["Code APE"].nunique())
```

```
Nb of possible legal forms:  160
Nb of possible business types:  1055
```

There are many of those in this dataset:
- 160 different legal forms
- 1055 different activity types





With regards to the legal form, 4 of those make the most of the data set and the remaining 156 can likely be grouped into one category.

With regards to the business types, although a few codes are well represented the vast majority makes only a small fraction of the data set. By looking on the Internet, one can also read from those codes (5 characters) one can extract a more high level type of activity, which I will do in the data processing.

2.2 "Radiation" information

This data set contains the list of all companies that were radiated from the commercial registry between January 2019 and October 2019.

Size:
This dataset contains ca. 120 thousands records, and for each of those 29 features / columns.

```
# Size of data file
radiations_2019.shape
```

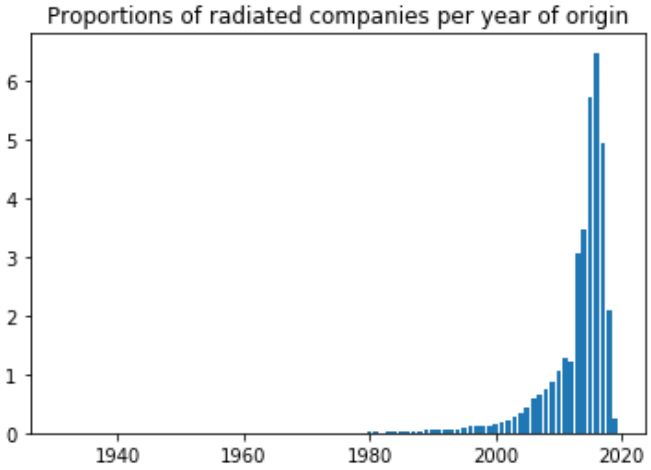
```
(120463, 29)
```

Columns / features:
Many columns are identical to the 1st dataset. In this dataset we are only interested in the company ID and the "Date radiation" which states when the company was officially removed from the commercial registry.

```
# Columns / feature names
radiations_2019.columns
```

```
Index(['Dénomination', 'Siren', 'Nic', 'Forme Juridique', 'Code APE',
       'Secteur d'activité', 'Adresse', 'Code postal', 'Ville', 'Num. dept.',
       'Département', 'Région', 'Code Greffe', 'Greffe',
       'Date immatriculation', 'Date radiation', 'Statut', 'Geolocalisation',
       'Date de publication', 'Nom commercial', 'Date immatriculation origine',
       'Sigle', 'Devise', 'Durée', 'Date cloture 1er exercice',
       'Date arreté des comptes', 'Etat', 'etat_pub', 'fiche_identite'],
      dtype='object')
```

For the exploration phase, I was also interested in analysing if predominantly young companies go out of business or if it was not the case. From the graph below one can see that (apparently, as we do not have yet the comparison with the population age) rather young companies go out of business then older ones.



3. DATA PRE-PROCESSING

Before being able to build, test and evaluate a model, the data needs to be cleaned and pre-processed. As seen from the exploration phase, I will need to:

- Extract relevant features
- Deal with many NaNs
- Convert to right datatype
- Merge both financial and radiations datasets
- Do some feature engineering (incl. hot encoding)
- Save the data into "cleaned" csv files

3.1 Deal with NaNs

Some of the features with many NaNs (see above) have not much importance for this analysis and have been dropped out (in my case those are not in the "col_to_keep" array).

```
col_to_keep = ['Dénomination', 'Siren', 'Forme Juridique', 'Code APE', 'Num. dept.', 'Date immatriculation',
               'Date de publication', 'Date de cloture exercice 1', 'Date de cloture exercice 2',
               'Date de cloture exercice 3', 'CA 1']
```

```
# Retain columns for pre-processing
df = df.loc[:, col_to_keep]
```

Unfortunately some of those features with many NaNs, like "CA 1" (turnover in year 1), "Resultat 1" (net result in the year 1), would have likely helped me a lot in building up this model. As the share of Nas is too high (around 20% and more), I have decided not to use those directly.

However I will use this information to build another feature: whether or not each company has actually really published its turnover in year 1 (e.g. the last year, in this case 2017). My hypothesis is that amongst others, companies in bad shape and those that are not required by law to disclose a number (essentially small companies) do not publish and hence this information could be used in the model.

```
# Record in new column if financial information (turnover) was published on last year
df['publish'] = df['CA 1'].isna().astype(int)
df.drop("CA 1", axis = 1, inplace = True)
```

Once all columns with Nas are removed and the information above coded, I have removed all rows with at least one Na, as with this I didn't lose many data points compared to the total number of records.

```
# Drop all rows with at least one NA; I still keep most of the information available
df.dropna(axis = 0, how = 'any', inplace = True)
```

3.2 Convert to right data type

Essentially 2 types of features need to be corrected:

- Dates features into date formats
- Company code ("Siren") into integer to be able to merge both data sets

Dates:

```
col_dates = ['Date immatriculation', 'Date de publication', 'Date de cloture exercice 1',
         'Date de cloture exercice 2', 'Date de cloture exercice 3']
```

```
# Convert date columns into dateformat
for col in col_dates:
    df[col] = pd.to_datetime(df[col])
```

Company code:

```
# Convert Siren into integer to merge with other information
df.Siren = df.Siren.astype(int)
```

3.3 Merge datasets

Once the "Siren" column in both datasets has the same data type, I merged both. As I will not use the exact date of radiation but "only" the information whether or not this company went out of business, I created a feature called "rad": 1 if the company was radiated and 0 if not.

```
df_rad = df_rad.iloc[:, 1:3]
df_merge = df.merge(df_rad, how = 'left', on = 'Siren')

# indicate radiation
df_merge["rad"] = 1 - df_merge["Nic"].isna().astype(int)

# remove last not interesting column
df_merge.drop("Nic", axis = 1, inplace = True)
```

3.4 Dimensionality reduction of categorical features

We saw in the data exploration part that 2 categorical features had quite a few possible outcomes:
- legal form
- Code APE

In this section I will reduce the number of possible outcomes of those 2, while keeping (I believe) the essential information.

Legal form:

We saw in the exploration part that the top 4 possible values are by far the most represented and that the remaining 156 are quite "rare" (in proportions). I have then decided to keep only 5 possible outcomes for this feature.

```
# Keep only the most important legal forms
legal_form_to_keep = df_merge.groupby("forme_juridique")["age"].count().sort_values(ascending = False)[:4]
legal_form_to_keep = np.array(legal_form_to_keep.index)
df_merge["legal_form_simple"] = df_merge["forme_juridique"].apply(lambda x: x if x in legal_form_to_keep else "others"
df_merge.drop(["forme_juridique"], axis = 1, inplace = True)
```

Code APE:
By looking on the Internet, one can read that this categorical feature can be summarized into a higher level by taking only the 1st two characters of this code.

```
# Transform business type into a new APE code
df_merge.code_ape = df_merge.code_ape.astype(str)
df_merge["new_ape"] = df_merge["code_ape"].apply(lambda x: str(x)[:2])
df_merge.drop(["code_ape"], axis = 1, inplace = True)
```

3.5 Feature engineering

In the previous part we already created 2 new features:
- "rad": whether or not the company was radiated in 2019
- "publish": whether or not the company actually disclosed its turnover in 2017

Now we will "engineer" 2 additional features:
- delay: the number of days between the end of the fiscal year and the actual publication of those results in 2017
- age: age in years of the company in 2019 (when the radiations are known)

Delay:
Motivation: my hypothesis is that when companies are in bad shape the auditing of the results takes longer than usual and/or the companies publish those at the latest legally possible date to avoid giving this information public to commercial partners.

```
# Calculate delay between end of fiscal year and date of publishing
df["delay"] = df["Date de publication"] - df["Date de cloture exercice 1"]
df.delay = df.delay.apply(lambda x: int(x.days))
```

Age:
Motivation: as we saw in the exploration phase, companies going out of business seem to be rather young companies. Coding this information will likely help us in the model.

```
# Calculate age of company in 2019 when radiations are recorded
df['yob'] = df["Date immatriculation"].apply(lambda x: x.year)
```

```
# Create a feature with the age in years of the company
df_merge["age"] = df_merge["yob"].apply(lambda x: int(2019 - x))
df_merge.drop("yob", axis = 1, inplace = True)
```

3.6 Hot encoding

The categorical features still need to be hot encoded before saving the data and being able to feed it into a machine learning pipeline. Of all the 3 possible years in the financial information, I have decided to keep only the last one (e.g. 2017). As a consequence there remains only 3 categorical features to hot encode:

- num_dept: geographical area
- new_ape: simplified activity code
- legal_form_simple: simplified legal form

```
# Hot encode data
df_final = pd.get_dummies(df_merge, columns=['num_dept', 'new_ape', 'legal_form_simple'],
                          drop_first = True, prefix=['num_dept', 'new_ape', 'legal_form_simple'])
```

3.7 Saving

The last part of the ETL pipeline is in my case to save the cleaned data frame into a new csv file, ready for further parts of this project.

```
# Load data into a new CSV
df_2017_19.to_csv("cleaned_data_2017.csv", sep = ";")
```

3.8 Conclusion / Discussion of ETL part

The final documented steps look deceptively simple. In the absence of a proper document of all features, it required quite a few hours to bring it together and some feedback loops of the machine learning part to tune some of those and create / delete some features.

The nature of the features is in itself not complicated, the proportions of NaNs and the total number of records made it for me for complicated than it actually is / looks to be.

4. MACHINE LEARNING

Now is time to actually build up a model to predict which companies are likely to go out of business.

4.1 Load data

Using the cleaned data of part 1, I need to load this data into the new notebook using the following function.

```
## Quick function to load the data and make the few transformations
def load_transform(filepath):
    '''
    Quick function to load the data and drop one column
    INPUT:
    filepath of data to be loaded
    OUTPUT:
    dataframe ready for ML Pipeline
        '''

    # load dataframe
    df = pd.read_csv(filepath, sep = ";")

    # drop column coming from saving date
    df.drop("Unnamed: 0", axis = 1, inplace = True)

    return df
```

```
# Load 2017 data
df = load_transform("cleaned_data_2017.csv")
```

4.2 Prepare data

From the loaded data, I now need to extract X and y, as well as split into a training and a test set.

```
# Function to prepare training and testing set
def prep_train_test(df, test_size = 0.25):
    '''
    Extract X and y, Split data into training and testing set
    INPUT:
    df: dataframe
    test_size (float): share of testing set
    OUTPUT:
    X_train, X_test, y_train, y_test
    '''

    # Extract X and y
    y = df['rad'].values
    X = df.drop(['rad', 'siren'], axis = 1).values

    # Split training and testing set
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = test_size, random_state = 1)

    return X_train, X_test, y_train, y_test
```

```
# Prepare training and testing set
X_train, X_test, y_train, y_test = prep_train_test(df)
```

4.3 Select Model / Classifier

Before trying to optimize a classifier type, I wanted to test out which classifier works best on this data. For this I trained several classifiers and tested their performance on the test set.

To be able to compare the performance amongst classifiers but also against the baseline, I calculated the basic accuracy and precision of an (ultra) simplified assumption, when predicting no company go out of business.

Details about the function can be found in the Notebook DSND-CapstoneProject_Part_2

In this function I also used a StandardScaler object to scale my features as some are between 0 and 1, and some (like delay can reach 200 to 300).

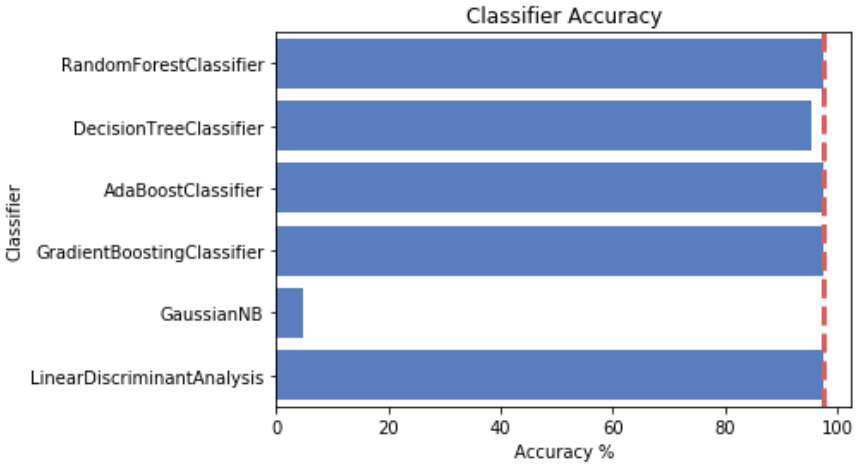
Baseline accuracy and precision:

```
# Compute basis performance
acc_basis = 1-y_train.mean()
prec_basis = 1-acc_basis
print("Basic accuracy without model on the training set: {} %".format(np.round(acc_basis * 100,3)))
print("Basic precision without model on the training set: {} %".format(np.round(prec_basis * 100,3)))
```

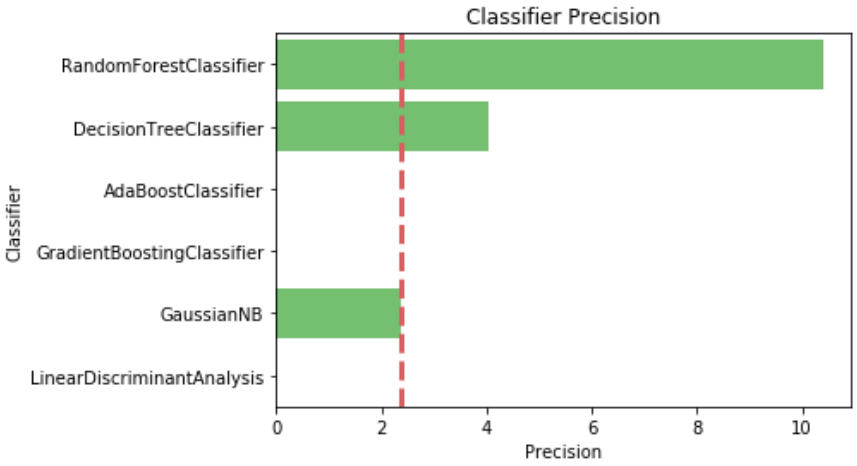
```
Basic accuracy without model on the training set: 97.633 %
Basic precision without model on the training set: 2.367 %
```

Nota: "training set" should read "test set" here, the naming in the print function did not match the computation made.

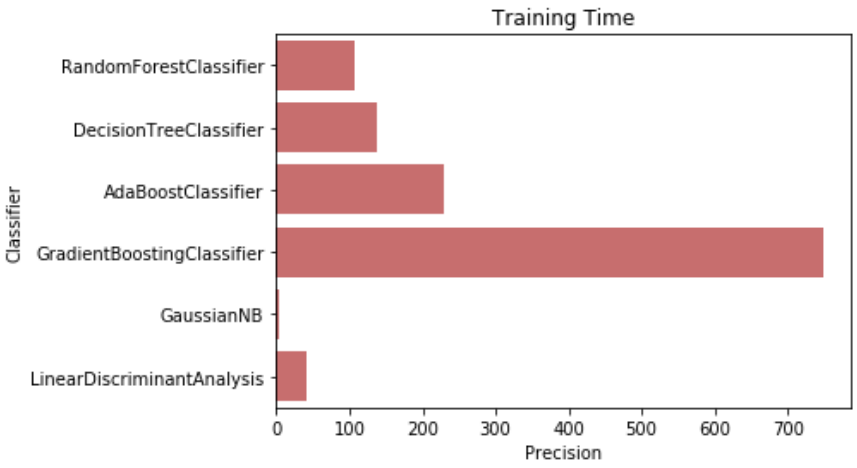
Accuracies of the models (red line = baseline accuracy):



Precisions of the models (red line = baseline accuracy):



Training calculation times of the models:



Conclusion:

Using the default classifiers hyperparameters, the RandomForestClassifier is by far the best model for our purpose:

- Precision well above all other models and also well above the baseline precision
- One of the quickest model to train

As a consequence, I have decided to use a RandomForestClassifier to go further on in this project.

4.4 Optimize Classifier

Using a RandomForestClassifier, I then used Grid Search to optimize for the hyperparameters.

GridSearch applied on this data set with my computer ended up being very, very long. The final set of hyperparameters after having tried a few of those is below:

```
# Define the pipeline model with a scaler and a RandomForestClassifier
model = Pipeline([
    ('scale', StandardScaler()),
    ('clf', RandomForestClassifier())
])
```

```
# New test to try on precision
param_grid_new = {"clf__n_estimators" : [10,20],
                  "clf__criterion": ["gini", "entropy"],
                  "clf__max_depth": [None, 5],
                  "clf__bootstrap": [True, False],
}
```

Importantly in my case I need to optimize by “precision”, otherwise the optimized model on accuracy has a precision of 0, which does not fulfil the objective of this project.

```
# Instantiate GridSearchCV
model_final_prec = GridSearchCV(model, param_grid_new, verbose = 5, cv=2, scoring = 'precision')
model_final_prec.fit(X_train, y_train)
```

Results of the GridSearchCV:

```
# Print out parameters of best model to avoid having t re-run the exercise
print("Parameters of the best model are: ",model_final_prec.best_params_)
```

```
Parameters of the best model are:  {'clf__bootstrap': True, 'clf__criterion': 'entropy', 'clf__max_depth': None, 'clf
__n_estimators': 20}
```

One very important hyperparameter seems to be max_depth. Any number (at least below 5) does not allow a precision more than 0. This algorithm needs to build "deep" trees to catch the few positives (in relation to the negatives) in the training set.

4.5 "Understand" Classifier

Using a few methods, we can better understand which features are important.

```
# Extract feature importance to analyse which feature is most relevant
importances_new = pd.DataFrame(model_final_prec.best_estimator_.named_steps["clf"].feature_importances_)
```

The top 10 features by importance make 86% of the total importance in this model.

```
# Analyse which features are most important (sum is equal to 1)
feature_importance_new = feature_importance_new.sort_values(by = "weight", ascending = False)
```

```
# How many make the most of the weight
feature_importance_new[:10].sum()
```

```
weight    0.861987
dtype: float64
```

In the top 10 we find the age of the company, a legal form and the delay.

```
feature_importance_new[:10]
```

	weight
age	0.225692
legal_form_simple_Societe par actions simplifiee a associe unique	0.151295
delay	0.141137
new_ape_47	0.090943
new_ape_64	0.063907
legal_form_simple_Societe par actions simplifiee	0.054712
new_ape_43	0.043968
new_ape_62	0.038121
num_dept_49	0.027228
num_dept_35	0.024984

The top 3 features make ca. 50% of the total importance, 2 of which are features we engineered for this project.

5. RESULTS & DISCUSSION

5.1 Test model on test set

Now let's apply the model on the test set and compute accuracy and precision:

```
# Predict on test set
y_preds = model_final_prec.predict(X_test)
```

```
# Get performance on test set
print("Precision on test set: {} %".format(np.round(precision_score(y_test, y_preds)*100,3)))
print("Accuracy on test set: {} %".format(np.round(accuracy_score(y_test, y_preds)*100,3)))
```

```
Precision on test set: 13.002 %
Accuracy on test set: 97.439 %
```

In order to assess the performance of the model, the basic accuracy and precision on the test set is also computed:

```
# Compute performance lift-off vs basic assumption
print("Basic Precision on test set: {} %".format(np.round(y_test.mean()*100,3)))
print("Basic Accuracy on test set: {} %".format(np.round((1-y_test.mean())*100,3)))
```

```
Basic Precision on test set: 2.349 %
Basic Accuracy on test set: 97.651 %
```

Combining both, we can compute the performance lift-off of the model compared to the basic assumption.

```
# Lift off
perf_lift_off_prec = precision_score(y_test, y_preds) / y_test.mean() - 1
perf_lift_off_acc = accuracy_score(y_test, y_preds) / (1-y_test.mean()) -1

print("Performance lift-off in terms of precision: {} %".format(np.round(perf_lift_off_prec*100,1)))
print("Performance lift-off in terms of accuracy: {} %".format(np.round(perf_lift_off_acc*100,1)))
```

```
Performance lift-off in terms of precision: 453.5 %
Performance lift-off in terms of accuracy: -0.2 %
```

On the test set of the 2017 financial information and 2019 radiations, this model has an excellent precision of ca. 13% (1 in 8 companies potentially contacted) vs a basic precision of ca. 2% (1 in 50 companies potentially contacted).

5.2 Test model on previous year

The aim of this project is not only to be able to build up a model functioning well for one year, but also to be able to apply it on a set of data and actually contact companies to try buying them. For this I need this model to be also good enough when applied to other years, e.g. to see if it generalizes well not only on the test set of 2017 but also apply on another year.

To test this I loaded the 2016 data (2016 financial information and 2018 radiations) and applied the model saved as a pickle file.

Load model and data:

```
# Load model
pkl_filename = "pickle_model_2017-19.pkl"
with open(pkl_filename, 'rb') as file:
    pickle_model = pickle.load(file)
```

```
# Test on 2016 financial information (and 2018 so called radiations)
df_2016 = load_transform("cleaned_data_2016.csv")
```

```
# Extract X and y (across the entire set, the entire set is now a test set)
X_2016 = df_2016.drop(['siren', 'rad'], axis = 1)
X_2016['age'] = X_2016['age'].apply(lambda x: x-1)
y_2016 = df_2016['rad']
```

Make predictions on 2016 data:

```
# Predict using model
y_preds_2016 = pickle_model.predict(X_2016)
```

Test model on 2016 data:

```
# Test on entire set
print("Precision on 2016 data is then: {} %".format(np.round(precision_score(y_2016, y_preds_2016)*100,3)))
print("Accuracy on 2016 data is then: {} %".format(np.round(accuracy_score(y_2016, y_preds_2016)*100,3)))
```

```
Precision on 2016 data is then: 2.875 %
Accuracy on 2016 data is then: 97.119 %
```

Compare with baseline performance:

```
# Compute performance lift-off vs basic assumption
print("Basic Precision on test set: {} %".format(np.round(y_2016.mean()*100,3)))
print("Basic Accuracy on test set: {} %".format(np.round((1-y_2016.mean())*100,3)))
```

```
Basic Precision on test set: 2.667 %
Basic Accuracy on test set: 97.333 %
```

Compute performance lift-off:

```
# Lift off
perf_lift_off_prec_2016 = precision_score(y_2016, y_preds_2016) / y_2016.mean() - 1
perf_lift_off_acc_2016 = accuracy_score(y_2016, y_preds_2016) / (1-y_2016.mean()) - 1

print("Performance lift-off in terms of precision: {} %".format(np.round(perf_lift_off_prec_2016*100,1)))
print("Performance lift-off in terms of accuracy: {} %".format(np.round(perf_lift_off_acc_2016*100,1)))
```

```
Performance lift-off in terms of precision: 7.8 %
Performance lift-off in terms of accuracy: -0.2 %
```

Discussion:
The performance of the model trained on the 2017 data is relatively poor on the 2016 data, which tends to indicate that this model does not generalize to other years.

5.3 Build model on 2016 data

In order to see where the differences can come from, I decided to build the same model not on the 2017 data but on the 2016 and explore differences in the model.

Important for me was to assess if the same features are important in both models.

Below are the top 10 features of the model built on the 2016 data by importance:

```
feature_importance_new[:10]
```

	weight
delay	0.486259
age	0.218251
publish	0.024598
legal_form_simple_Societe a responsabilite limitee a associe unique	0.015135
legal_form_simple_Societe par actions simplifiee	0.007152
new_ape_68	0.006181
legal_form_simple_Societe par actions simplifiee a associe unique	0.005902
new_ape_46	0.005218
num_dept_75	0.005046
new_ape_43	0.004949

By comparing both one can see that amongst others the activity codes differ and the importance of the similar features in both top 10 lists are quite different.

This 2016 model also performed quite poorly on the 2017 data with a precision just 0.1% better than the baseline precision.

```
# Test on entire set
print("Precision on 2016 data is then: {} %".format(np.round(precision_score(y_2017, y_preds_2017)*100,3)))
print("Accuracy on 2016 data is then: {} %".format(np.round(accuracy_score(y_2017, y_preds_2017)*100,3)))
```

```
Precision on 2016 data is then: 2.468 %
Accuracy on 2016 data is then: 97.292 %
```

```
# Compute performance lift-off vs basic assumption
print("Basic Precision on test set: {} %".format(np.round(y_2017.mean()*100,3)))
print("Basic Accuracy on test set: {} %".format(np.round((1-y_2017.mean())*100,3)))
```

```
Basic Precision on test set: 2.363 %
Basic Accuracy on test set: 97.637 %
```

```
# Lift off
perf_lift_off_prec_2017 = precision_score(y_2017, y_preds_2017) / y_2017.mean() - 1
perf_lift_off_acc_2017 = accuracy_score(y_2017, y_preds_2017) / (1-y_2017.mean()) - 1

print("Performance lift-off in terms of precision: {} %".format(np.round(perf_lift_off_prec_2017*100,1)))
print("Performance lift-off in terms of accuracy: {} %".format(np.round(perf_lift_off_acc_2017*100,1)))
```

```
Performance lift-off in terms of precision: 4.4 %
Performance lift-off in terms of accuracy: -0.4 %
```

5.4 Alternative models

As the model built does not generalize well on other years, I decided to test if a more general model (eg. With fewer features) could generalize better to other years. The idea was to test if losing in precision in a given year could help being more performing on another.

This test can be seen in the notebook `CapstoneProject_Part_3`

Unfortunately this model didn't provide much useful information / applicable model as the performance of the model on another year was a gain of just 0.5% Pt.

6. CONCLUSION

Conclusion and potential further improvement steps:

The classifier built was astonishingly very performing for any given year. Such a precision in real life would be a game changer in the private equity / restructuring field.

Unfortunately as we saw both models built for any given year didn't work well on other years, which tend to indicate (as could be expected) that there are time series element to be taken into account. With the data at hand (only 2 years available) one cannot further analyse this time component.

In order to improve this model and make it potentially useful in real life, I would like to try two new approaches:

- Training a neural network on the 2016 data and test it on the 2017 one. Maybe could a neural network spot patterns that a "standard" classifier does not
- Using / constructing additional features (for example on the 2 other fiscal years that are also recorded in the financial information data

Personal reflection:

Watching videos and completing projects along the way was very helpful for me in understanding the Data Science process and its tools. Working on a "live" project on my own without specific online help pushed me further and enabled me to learn to really apply what I learned along the way.

The data engineering part of the process takes a lot of time for what seems in the end very simple steps. With this data and optimizing on the precision, I was confronted with the importance of some hyperparameters (max depth on the RandomClassifier). Choosing the "wrong" ones lead to a precision of 0, and it took me quite some time to understand that it played such an important role.

For my next project I will try to "think more before doing / testing" as I believe I lost dozens of hours due to not enough structuring my thoughts and the process upfront.

I enjoyed a lot this project and will definitely try going further with improving this one and starting new projects. Considering my background (business), I can't work on topics that are too complicated on a programming perspective. I can "compensate" with subject matter knowledge in the topics I'm interested in.