Machine Learning Engineer Nanodegree

CAPSTONE PROJECT

Kennedy Sousa
December 11th, 2017

Definition

Project Overview

Bank ACME's[1] Office of Compliance is a department responsible for monitoring the activities and conduct of employees: whenever an irregularity is detected, the bank must analyze whether the irregularity stems from misconduct or weaknesses in the process, in order to mitigate the operational risk and apply the penalty to those involved, if applicable, including possible compensation for financial losses.

The procedure starts with a process called *preliminary analysis* that consists in an investigation and aims to gather information about the issue, like authorship, which rule was broken, description of the facts, value involved, etc. After all the relevant information is gathered, the final report and the chain of evidence are sent to decision-making authority for deliberation. If the case is admitted, the indictee becomes defendant and is subject to penalties like written reprimand, suspension and discharge.

This project addresses the real world problem of identifying whether the case will be admitted or not, based in some multiple-choice fields filled in the report. The architecture of the project is mostly based on Finding Donor's project (Udacity)[i], which has a very similar proposal – using a binary classifier to predict the output. In fact, some code was reused and *visuals.py* was used in its entirety, with some adaptations.

The following links provide solutions to similar problems:

https://github.com/udacity/machine-learning/tree/master/projects/finding_donors

https://datascienceplus.com/building-a-logistic-regression-in-python-step-by-step/

http://nbviewer.jupyter.org/gist/justmarkham/6d5c061ca5aee67c4316471f8c2ae976

Problem Statement

The goal is to create a basic API that receives an input (the observations) and returns the predicted class (process admitted or not); the tasks involved are the following:

1. Download and preprocess the labeled data
2. Create a naïve classifier
3. Train an estimator
4. Test the estimator against the naïve classifier

[1] Some names and identifying details have been changed.

5. Improve the estimator
6. Create a web service to deliver the functionalities

The final application is expected to give a clue about decision trend.

Evaluation Metrics

The goal of the ML model is to learn patterns that generalize well for unseen data instead of just memorizing known data as its target, i.e. overfitting to training data.

Typical metrics applied to binary classifiers are accuracy, precision, recall, and F1-measure. Each metric measures a different aspect of the predictive model. Accuracy measures the fraction of correct predictions. Precision measures the fraction of actual positives among those examples predicted as positive. Recall measures how many actual positives were predicted as positive. F1-measure is the harmonic mean of precision and recall.

Accuracy	**Precision**	**Recall (sensitivity)**
$\frac{(TP+TN)}{(TP+TN+FP+FN)}$	$\frac{TP}{(TP+FP)}$	$\frac{TP}{(TP+FN)}$

In order to check if the model is performing well on unseen examples it takes into account accuracy and Fβ-score that considers both precision and recall:

$$F_\beta = (1+\beta^2).\frac{precision\ .recall}{(\beta^2.precision)+recall}$$

Both metrics get a score in the range [0, 1].

The module *sklearn.metrics* provides the function *accuracy_score* to compute the accuracy, either the fraction (default) or the count (normalize=False) of correct prediction. The function returns a value to maximize, the higher the better.

Considering the imbalance of the target labels (see Datasets and Inputs), for the purposes of the model, it was decided to adopt F-beta score weighted, which calculate metrics for each label, and find their average, weighted by support (the number of true instances for each label).

It is also important to keep track of the training and predicting time to ensure a good user experience.

Analysis

Data Exploration

The data comes from a corporate system in which the information collected about the occurrences are inserted. The dataset used in this project is a response to a data query sent to SQL Server and stored into a Comma Separated Values (.csv) file. Due to sensitivity of data all fields that could identify any person or entity were intentionally removed, like authorship, subject, etc. The dataset has a 61,016x39 shape and the following fields:

Feature	**Description**
nu_analise	Sequential number, by department and year, from 1 to 9999.
nu_unidade	Department responsible for analysis.

Feature	Description
dt_inicio_analise	Begin date.
nu_origem	Origin of register: 1. Audit 2. Denunciation 3. Manager 4. Credit Operation 5. Suspected money laundering 6. Guarantee Fund for Time Service 7. Robbery 8. Customer account refund 9. Office of Compliance
nu_unidade_ocorrencia	Department where the event occurred.
cd_situacao*	Analysis status: 1. Awaits filling 2. Awaits decision 3. Granted 4. Declined
ic_22	Are the supposedly irregular facts of behavioral order? 0. Not filled 1. Yes 2. No 3. Not identified
ic_23	Is there evidence of noncompliance with rules, laws or standards per employee, former employee or retiree? 0. Not filled 1. Yes 2. No 3. Not identified
ic_231	Has noncompliance determined the occurrence? 0. Not filled 1. Yes 2. No 3. Not identified
ic_24	Was the employee, former employee or retiree responsible for regulatory noncompliance identified? 1. Yes 2. No 3. Not identified
ic_25	Are there evidences of fraud or deceit in the conduct of the indictee? 0. Not filled 1. Yes 2. No 3. Not identified

Feature	Description
ic_26	Are there extenuating circumstances in the fact investigated? 0. Not filled 1. Yes 2. No 3. Not identified
ic_27	Are there aggravating circumstances in the fact investigated? 0. Not filled 1. Yes 2. No 3. Not identified
ic_28	Are there financial losses? 0. Not filled 1. Yes 2. No 3. Not identified
ic_281	Is there any possibility of future financial losses? 0. Not filled 1. Yes 2. No 3. Not identified
ic_282	Were the financial losses recovered or reimbursed? 0. Not filled 1. Yes 2. No 3. Not identified
nu_283_total	Amount lost (money).
nu_283_res	Amount recovered or reimbursed (money).
ic_284	Have the amounts been accounted? 0. Not filled 1. Yes 2. No
ic_29	Is there a third party involved? 0. Not filled 1. Yes 2. No 3. Not identified
ic_210	Are there signs of crime? 0. Not filled 1. Yes 2. No 3. Not identified

***** *Note that the 6th column from this dataset, 'cd_situacao', will be our target label (whether an analysis will become a disciplinary process or not). All other columns are features about each preliminary analysis in the database.*

Exploratory Analysis and Visualization

A cursory investigation of the dataset will determine how many registers fit into either group, and will tell us about the percentage of the analysis that became disciplinary process.

Total number of records: 54901
Analysis that became disciplinary process: 5985
Analysis archived: 48916
Percentage of disciplinary process admitted: 10.90%

A quick visual analysis (Figure 1) is very helpful to identify important insights, like the large number of features with answers marked as "not identified" - probably these features will not have the same importance as we expected. On the other hand, the indication of deceit and aggravating circumstances can be very useful when predicting the target. Although the signs of crime appear to be a determinant feature, there is always a chance of the employee be just a victim of criminals.

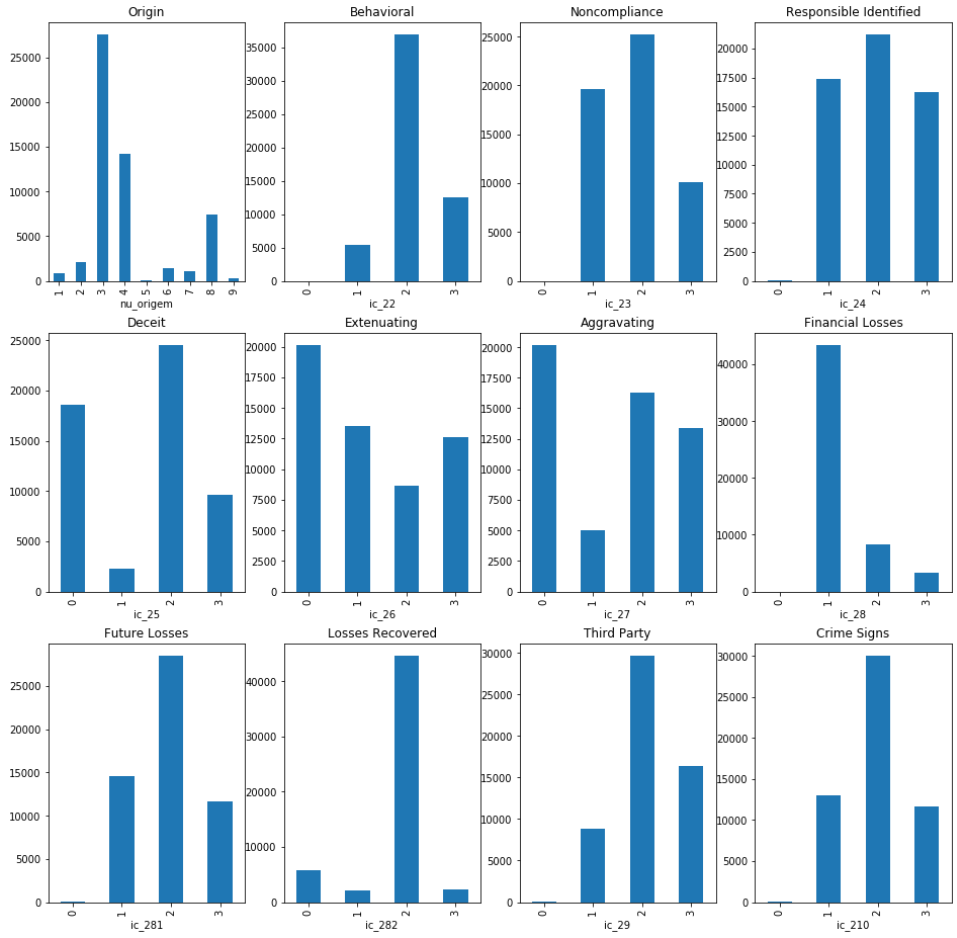


Figure 1 – Data exploration

Additionally, we chose a scikit-learn classifier (AdaBoost) that has a feature_importance_ attribute, which is a function that ranks the importance of features according to the chosen classifier. The result is detailed in the following plot:

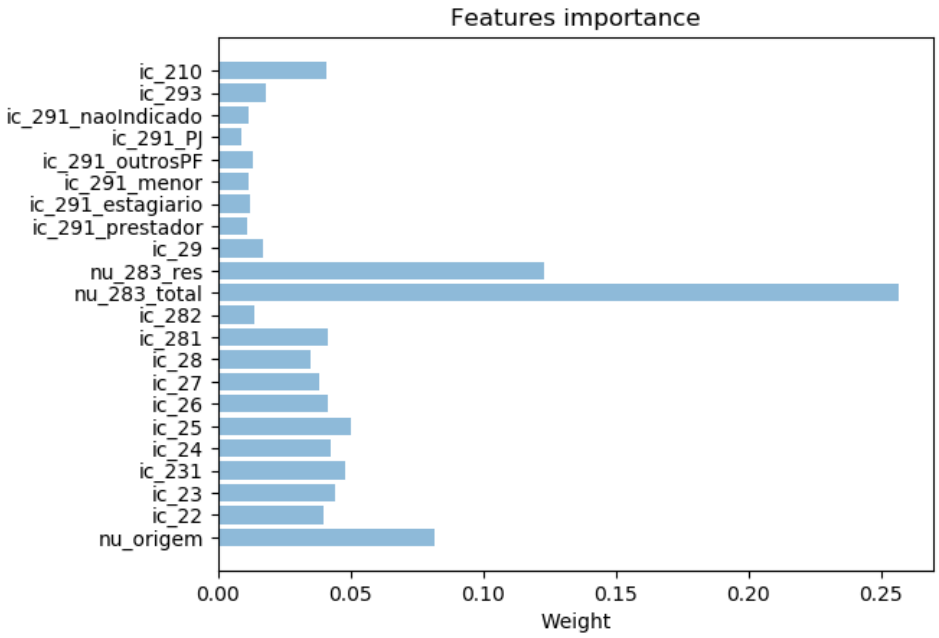


Figure 2 - Features Importance

Algorithms and Techniques

Classification is an example of pattern recognition and consists in one of the most commons problems in *Statistics* and *Machine Learning*. The goal is identifying to which of a set of categories a new observation belongs, based on a training set of data containing observations (or instances) whose category membership is known.

The following flowchart[ii] is useful to describe the approach used to solve the problem: the solution handles more than 50k samples and tries to predict a category based on labeled data.

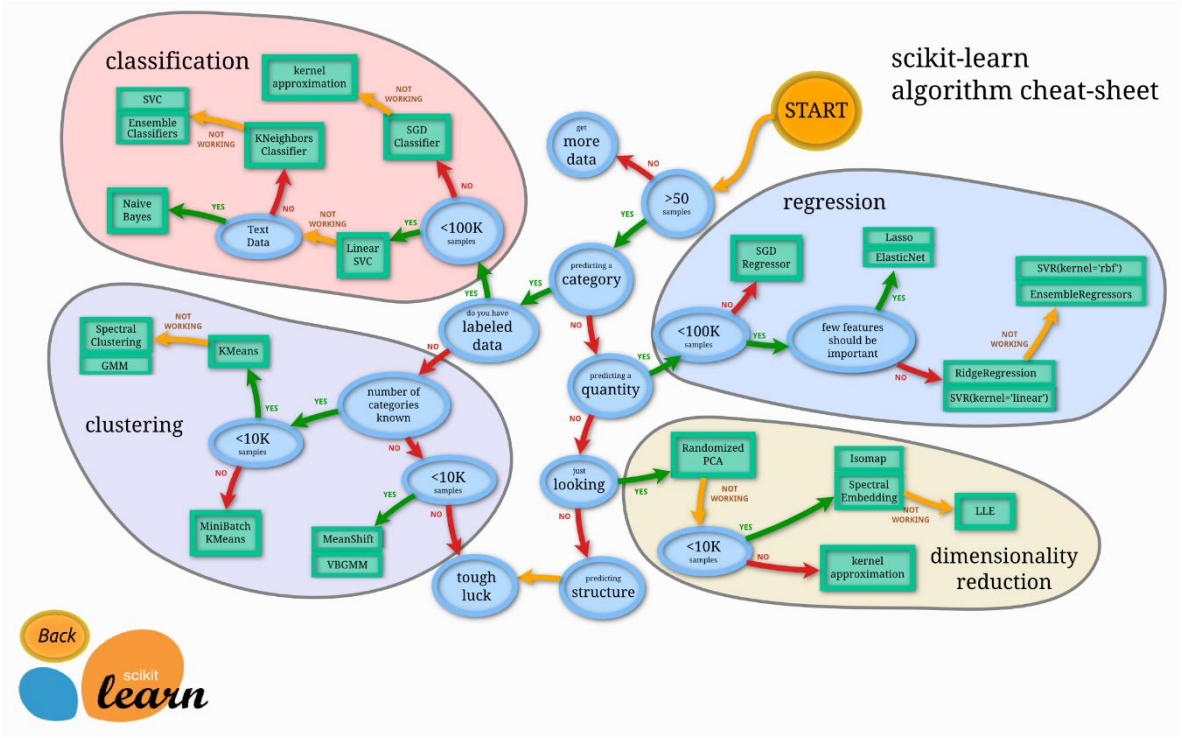


Figure 3 - Algorithms flowchart

Given the dataset features and the characteristics of the problem to solve, we considered three models as candidates: Gradient Boosting, Logistic Regression and AdaBoost.

Gradient Boosting

Gradient Boosting is a technique for regression and classification problems, which produces model in the form of an ensemble of weak prediction models, typically decision trees.[iii] Gradient Tree Boosting models are used in a variety of areas including web search ranking and ecology.[iv]

The advantages of Gradient Boosting are natural handling of data of heterogeneous features, predictive power, and robustness to outliers in output space.

On the other hand, the main disadvantage of Gradient Boosting lays on its scalability, due to the sequential nature of boosting it can hardly be parallelized.

Despite its low scalability, the Gradient Boosting model could be a good candidate because it has a good predictive power and it does not need to handle a large amount of information.

Logistic Regression

Logistic Regression is a regression model where the dependent variable is categorical - in the Preliminary Analysis case there is a binary dependent variable (granted or declined). Logistic Regression is used in various fields, including machine learning, most medical fields, and social sciences.[v]

Due to its simplicity, Logistic Regression is really fast and it has low variance which makes this model less prone to overfitting.

The Logistic Regression technique is useful, but it has limitations: it cannot predict continuous outcomes and it requires that each data point be independent of all other data points.[vi]

As said before, Preliminary Analysis dataset has a binary dependent variable and it has a large number of independent samples, which makes Logistic Regression a good option to solve the problem.

AdaBoost

The core principle of AdaBoost is to fit a sequence of weak learners (i.e., models that are only slightly better than random guessing, such as small decision trees) on repeatedly modified versions of the data. The predictions from all of them are then combined through a weighted majority vote (or sum) to produce the final prediction.[vii] Like others Ensemble Learning Methods, AdaBoost have been successfully used for face and objects detection.[viii]

The AdaBoost training process selects only those features known to improve the predictive power of the model, reducing dimensionality and potentially improving execution time as irrelevant features do not need to be computed. On the other hand, noisy data and outliers can negatively impact the prediction performance. Moreover, if a complex model is used as the base classifiers, this can lead to overfitting.[vii] Nevertheless, AdaBoost is still a good candidate because our dataset is preprocessed (clean, formatted, and structured) and it can be less susceptible to the overfitting problem than other learning algorithms in some problems.

Benchmark Model

The first benchmark created to test the model is a naïve classifier to make random guesses. It will establish a checkpoint in order to show how good can be use a ML model instead of just flip a coin and let it decide for us. The benchmark model achieved 0.1100 accuracy score and 0.1303 f-score. As the naïve classifier doesn't require training, it will not set precedents to training time.

Later, we will compare the naive predictor and three different algorithms then choose the best among them.

Finally, we will test different parameters for chosen model.

The expectation is keeping the training time bellow 5 seconds and predicting time bellow 500ms.

The two values above were determined by considering a reasonable response time for a web service in order to get a good user experience.

Methodology

Loading and preprocessing data

First of all, we must load all required libraries. After that, we need to load data from .csv file. By default, SQL Server 2012 uses semicolon (;) as separator when exporting query results. So, we should use the following code to get data loaded:

```
data = pd.read_csv(in_file, sep=';')
```

Before data can be used as input for machine learning algorithms, it often must be cleaned, formatted, and restructured — this is typically known as **preprocessing**. Fortunately, for this

dataset, there are no invalid or missing entries we must deal with, however, there are some qualities about certain features that must be adjusted. This preprocessing can help tremendously with the outcome and predictive power of nearly all learning algorithms.

At first, we discard all rows that do not have significant meaning for predicting outputs, like sequential and date, for instance.

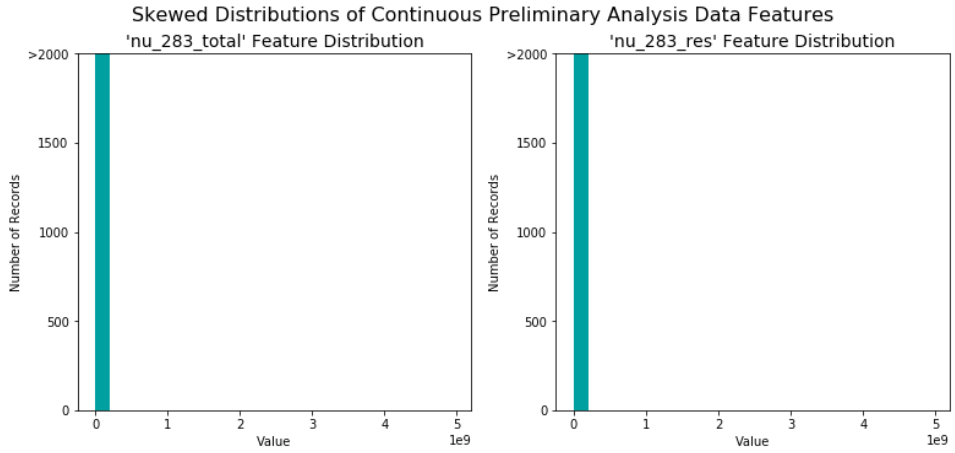
```
data = data.drop(['nu_analise', ..., 'dt_inicio_analise'], axis = 1)
```

Also, we must clean all 'NaN' occurrences of data frame and remove all the entries that have targets (cd_situacao) other than granted(3) or declined(4):

```
data = data.dropna(axis=0)

data = data[data.cd_situacao != 1]

data = data[data.cd_situacao != 2]

data = data[data.cd_situacao != 9]
```

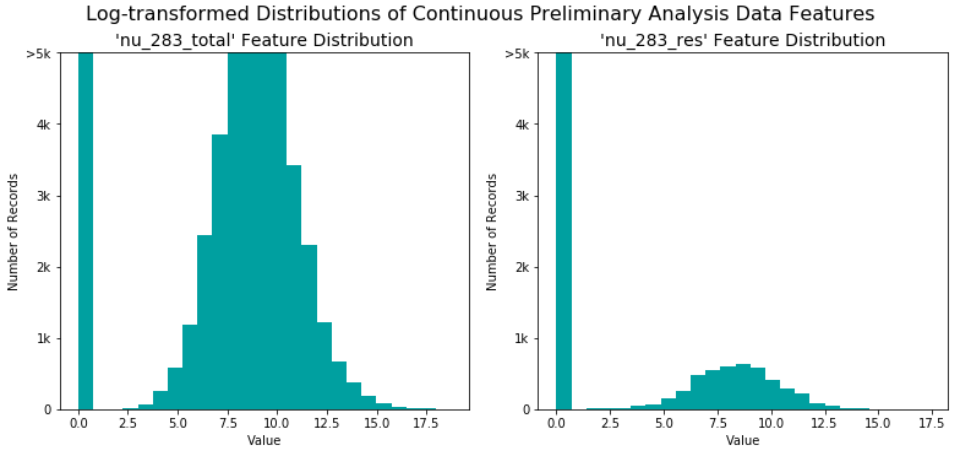
This dataset contains two features ('nu_283_total' and 'nu_283_res') whose values tend to lie near a single number (0.00), but will also have a non-trivial number of vastly larger than that single number. Algorithms can be sensitive to such distributions of values and can underperform if the range is not properly normalized.

	nu_283_total	nu_283_res
count	54899.00	54899.00
mean	63979.93	4368.25
std	1019804.91	227158.10
min	0.00	0.00
25%	236.06	0.00
50%	4597.09	0.00
75%	16315.23	0.00
max	135073856.00	34885571.12



The table and the histogram above shows that the values of both features are extremely skewed and tend to lie near 0.00.

In order to avoid problems in the algorithm performance due to highly-skewed features distributions in 'nu_283_total' and 'nu_283_res', we will apply a logarithmic transformation to reduce the range of values caused by outliers – this condition is clearly seen in 'nu_283_res' feature, where the upper percentile is 0.00 and the maximum value is 34,885,571.12. Since the logarithm of 0 is undefined, we must consider values slightly above when translating 0.

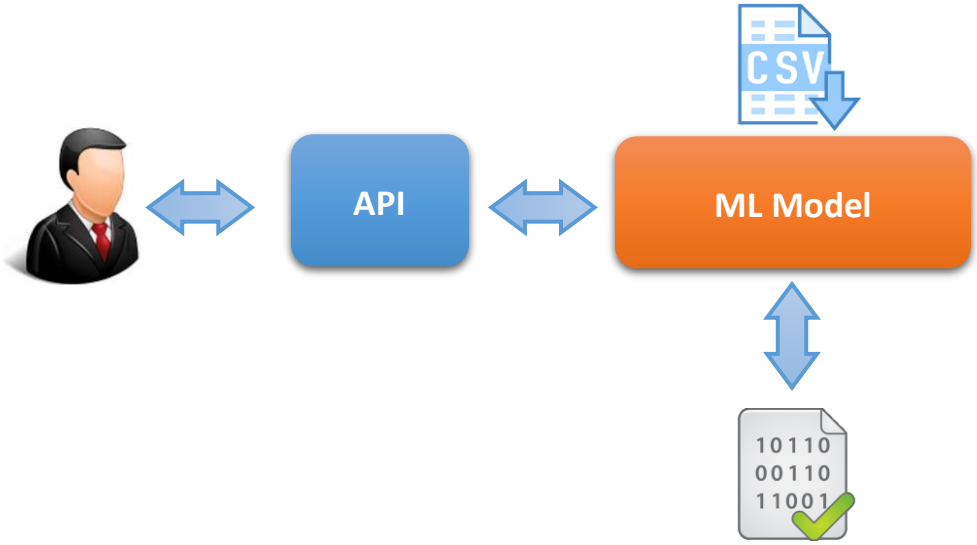


With clear data we now can store the target features in a different variable a remove it from original data frame:

```
outcomes = data['cd_situacao']

data = data.drop('cd_situacao', axis = 1)
```

With these steps done, the data is prepared to serve as input for a machine-learning model.

Methodology



The project consists in a machine-learning model that loads a comma separated values data file and stores the pre-trained model in a pickle file. If the pickle file is not present, the application runs the training and save the file. Then keep waiting the requests to make predictions and send the response.

Implementation

The implementation process can be split into three main stages:

1. Choose and implement the best classifier
2. The application development

Choosing the best classifier

During the first stage, the classifier is trained on the preprocessed training data. This stage can be subdivided into the following steps:

1. Split training and testing data:

```
X_train, X_test, y_train, y_test = train_test_split(data, outcomes,
test_size = 0.2, random_state = 0)
```

2. Create a naïve classifier to analyze the performance of random guessing:

We can do this just by running the following line of code and calculating accuracy and f-beta score:

```
outcome_predicted = outcomes.apply(lambda x:randint(0, 1))
```

The naïve predictor achieved 11% accuracy score and 13.03% fbeta-score.

3. Create the evaluation metrics to test the models. In this step, we evaluated the model against three subsets of data to analyze the behavior with different training sizes – in some cases the performance of the model can increase significantly as the size of training dataset increases. On the other hand, the time spent to train the model can be negatively impacted with more data, as we see in Gradient Boosting Classifier statistics above. If we observe this trend, we should consider reducing the dimensionality or choosing a model that performs better with large amount of data.
4. Create three models using Gradient Boosting Classifier, Logistic Regression, and AdaBoost Classifier. This step is definitely the most difficult and requires a deep knowledge or research about the models that can solve the problem, its pros and cons, and how it could be improved in the future. Fortunately, since the dataset is relatively small, the time spent to train the models was not a problem.

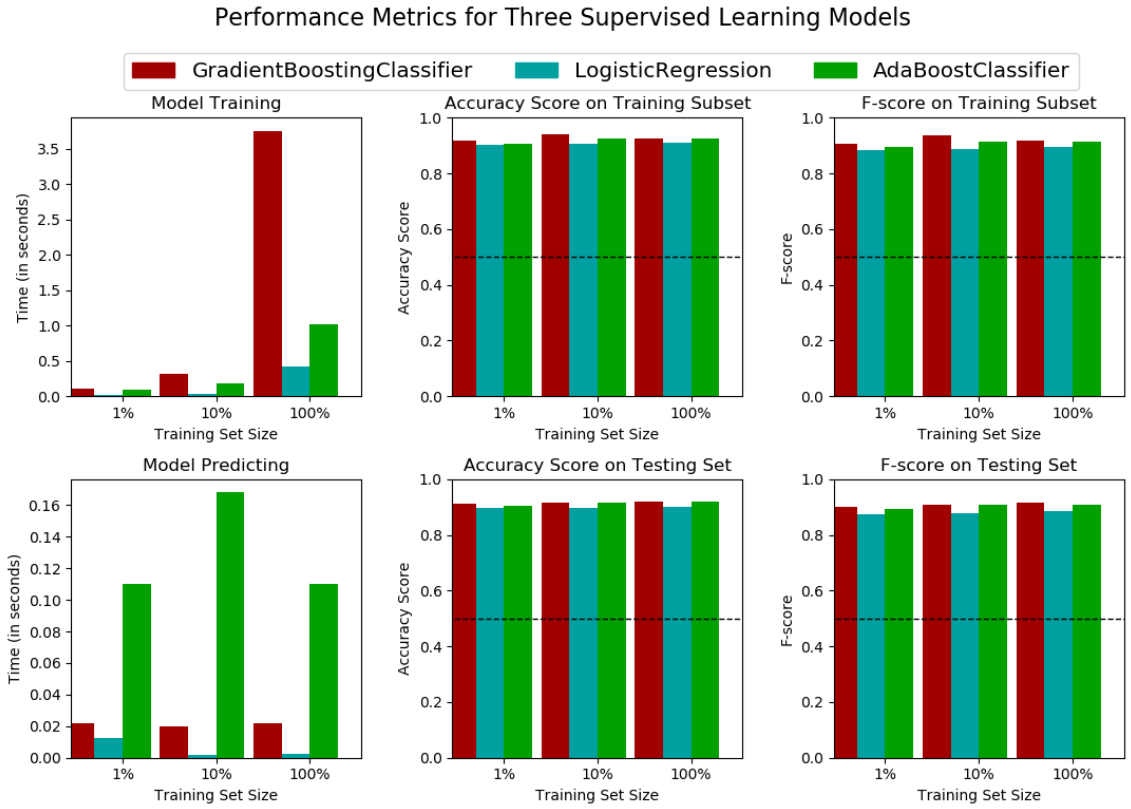
```
# Import the three supervised learning models

from sklearnfrom sklearn.ensemble import GradientBoostingClassifier

from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import AdaBoostClassifier

# Initialize the three models

clf_A = GradientBoostingClassifier(random_state=0)

clf_B = LogisticRegression(random_state=0)

clf_C = AdaBoostClassifier(random_state=0)
```

5. Test the models and collect the results

```
# Calculate the number of samples for 1%, 10%, and 100% of the
training datasamples_1 = len(X_train)/100

samples_10 = len(X_train)/10

samples_100 = len(X_train)/1

# Collect results on the learners

results = {}

for clf in [clf_A, clf_B, clf_C]:

      clf_name = clf.__class__.__name__

   results[clf_name] = {}

   for i, samples in enumerate([samples_1, samples_10, samples_100]):

      results[clf_name][i] = \

      train_predict(clf, samples, X_train, y_train, X_test, y_test)
```

```
# Run metrics visualization for the three supervised learning models
chosenvs.evaluate(results, accuracy, fscore)
```

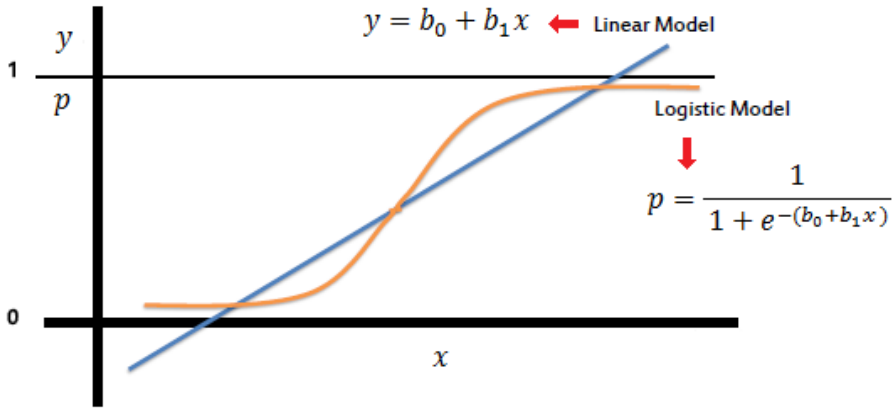
6. Choose the best model based on results



While GradientBoostingClassifier took almost 4 seconds to train with 30,000 samples, AdaBoostClassifier did the same 5x faster and LogisticRegression was even better, spending just a fraction of second both training and testing. Regarding the Accuracy Score, all the evaluated supervised learning models did a good job both in training and testing, with more than 80% accuracy: AdaBoostClassifier and LogisticRegression achieved almost the same result and GradientBoosting's score was slightly higher. Concerning F-score, GradientBoostingClassifier slightly outperformed the other models.

The best choice for Preliminary Analisys' case is **Logistic Regression** because it was very good in terms of accuracy and f-score, and it did this a lot faster than the other models.

Logistic regression makes use of one or more predictor variables (features) to predict binary dependent variables (in our case: declined or granted). To do that, logistic regression first takes the probability (p) of the event (y) happening for different levels of each independent variable (bi) and use a function called maximum likelihood estimation (MLE) to obtain the model coefficients that correlate predictors to the target, while keeping it into an acceptable range. To illustrate this, consider the graph below[ix]: the logistic function gives the line a "S" shape (orange line) to avoid estimating probabilities below 0 (zero) and over 1 (one), like occurs with linear regression (blue line).



To make predictions, we apply the logistic regression into the predictor variables: if the output is greater than a given cutoff value (like 0.5), the output is translated into 'granted'; else, the output is 'declined'.

Application development

In this stage we will work on the API module. The application should be able to:

1. Load pickle file

```
#  Load the saved model
print("Loading the model...")
loaded_model = None
with open('./models/'+clf,'rb') as f:
      loaded_model = pickle.load(f)
print("The model has been loaded... ")
```

2. Receive requests

```
try:
      request_json = request.get_json()
      test = pd.read_json(request_json, orient='records')
      analisys_id = test['cd_analise']
      test = test.drop('cd_analise', axis = 1)
except Exception as e:
   raise e
```

3. Make predictions

```
predictions = loaded_model.predict(test)

prediction_series = list(pd.Series(predictions))
```

4. Return answers

```
final_predictions = pd.DataFrame(list(zip(analisys_id, \
prediction_series)))

responses = jsonify( \
predictions=final_predictions.to_json(orient="records"))

responses.status_code = 200

return (responses)
```

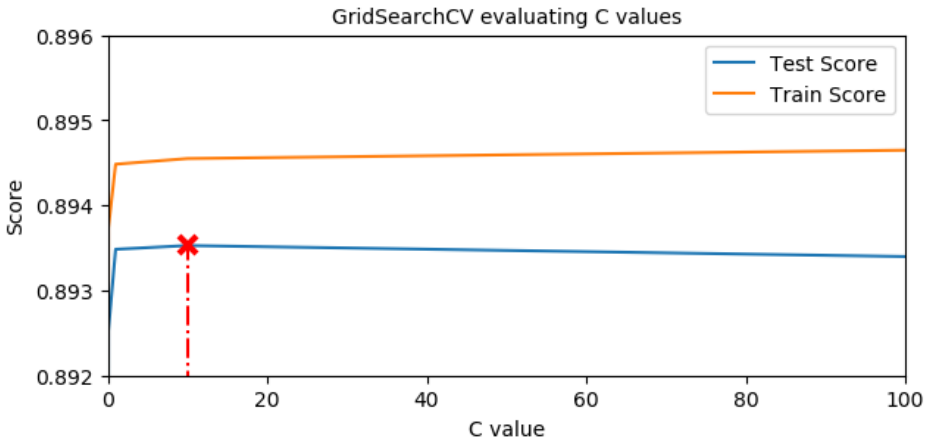
Refinement

Logistic regression, despite its name, is a linear model for classification rather than regression. In this model, the probabilities describing the possible outcomes of a single trial are modeled using a logistic function.

Due to its simplicity, Logistic Regression has only one main parameter to be tuned for binary classification – the parameter C, our regularization parameter. Being parameter C = 1/λ.

Lambda (λ) controls the trade-off between allowing the model to increase its complexity as much as it wants and trying to keep it simple. For example, if λ is very low or 0, the model will have enough power to increase its complexity (overfit) by assigning big values to the weights for each parameter. If, in the other hand, we increase the value of λ, the model will tend to underfit, as the model will become too simple.

With that said, we tried C = [0.001, 0.01, 0.1, 1, 10, 100] for the GridSearchCV in order to tune the model. The changes in the performance of Logistic Regression model were very subtle, however, we observed that C = 10 increases slightly the model performance, but it becomes prone to overfitting as C gets higher, decreasing accuracy and f-score. The following chart helps us to understand this phenomenon, with training score and testing score diverging as C increases to 100.



C	0.001	0.01	0.1	1	10	100
Test Score	0.8349	0.8854	0.8925	0.8934	0.8935	0.8933
Train Score	0.8361	0.8859	0.8937	0.8944	0.8945	0.8946
Rank	6	5	4	2	1	3

Therefore, we decided to keep the value of parameter C suggested by GridSearchCV in order to achieve best results.

Results

Model Evaluation and Validation

After performing all the tests, we can summarize the results in the following table

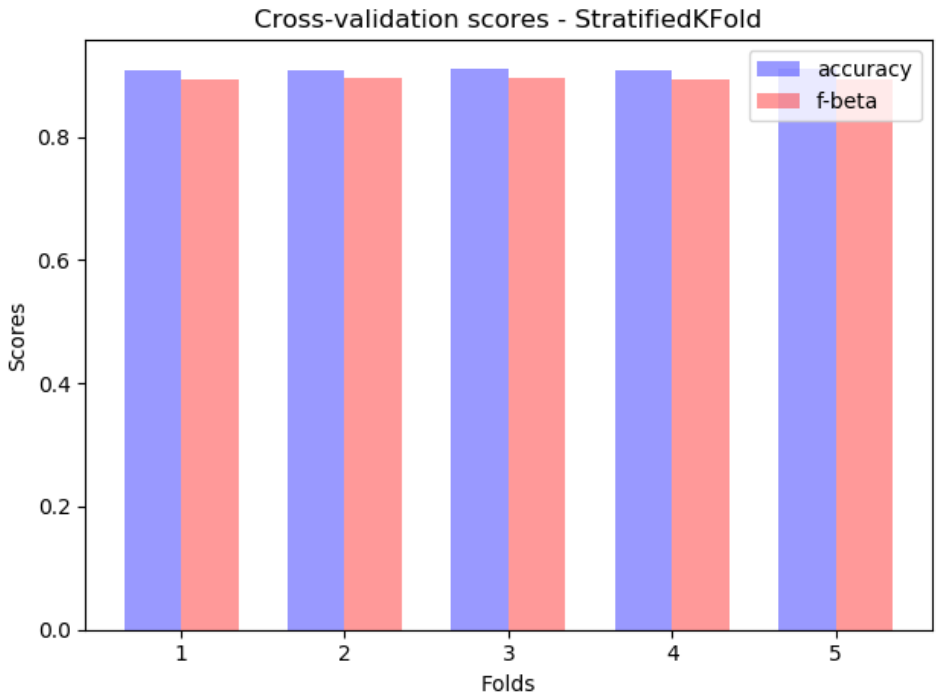
	Benchmark Predictor	AdaBoost	Gradient Boosting	Logistic Regression
Accuracy	0.1100	0.9183	0.9214	0.9025
F-Score	0.1303	0.9092	0.9142	0.8865
Training time	-	0.9545	3.1322	0.4359
Prediction time	-	0.1144	0.0225	0.0020

The Logistic Regression model was chosen because it performed quite well compared to AdaBoost and Gradient Boosting in terms of accuracy and f-score. Furthermore, due to its nature, Logistic Regression has low variance, which makes it very robust even with the default parameters. Thus, considering that the models were balanced, the determinant factor for deciding to adopt Logistic Regression was the processing time, both training and predicting.

Also, the model proved its reliability after computing cross-validated metrics. The *cross_val_score* helper function estimated the accuracy and f-beta score of the Logistic Regression algorithm by splitting the data, fitting a model and computing the scores 5 consecutive times (with different shuffled splits each time). The plot below demonstrates that the final model is unlikely to change, despite the changes in the dataset.

The following lines of code shows how the *cross_val_score* helper function was used. Note that the parameter 'cv' specifies the number of folds used. For binary classifier the StratifiedKFold is used.

```
fbeta_cv=cross_val_score(clf, X_train, y_train, cv=5, scoring=scorer)

accuracy_cv = cross_val_score(clf, X_train, y_train, cv=5,\
scoring='accuracy')
```



Finally, we can clearly see by the results summarized in the table above that the optimized Logistic Regression model outperforms the benchmark model both in accuracy (more than 8x better) and f-score (almost 7x better).

Justification

As users, we expect to get quick responses to our requests while accessing a website or consuming a web service.

Using Logistic Regression means that we will not have large delays in the backend of application.

- The accuracy is over 90%
- The f-score is over 85%
- The training time is below 500ms
- The prediction time is below 5ms

Finally, the application achieved its goal and is useful for the objective that it was designed for – providing a direction on how decisions are made in order to help resource planning and related issues.

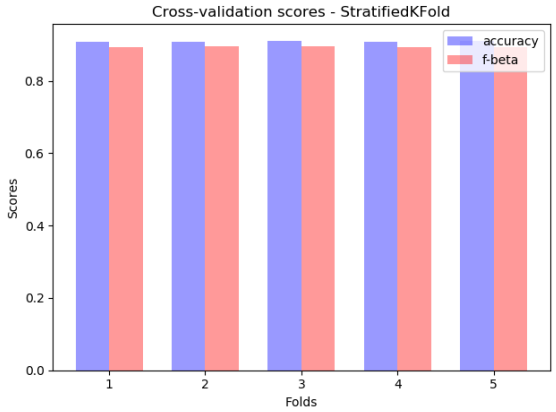
Conclusion

As deciding whether a preliminary analysis will become a disciplinary process or not is still discretionary and depends on a deeper observation of the facts, the main goal of the project was not predicting the right answer a 100% of the time. With about 90% accuracy and f-score it is possible to create a reliable strategy on how to handle the future disciplinary process, planning

resources, etc. Also, this makes it possible to avoid different decisions to similar issues - *ubi eadem ratio ibi eadem legis dispositio.*

Free-Form Visualization

```
Optimized Model
------
Final accuracy score on the testing data: 0.9023
Final F-score on the testing data: 0.8863

LogisticRegression(C=10, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=0, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```



The most important qualities about the chosen model are its robustness and reliability.

When we analyze the balanced results of the three different models tested, it becomes clear that all models succeeded in capturing the patterns in the dataset. The margin of error that occurred in the same intensity for all the models probably is related to discretionary characteristic of the decisions made. It is important highlight that the same issue can be faced in different ways depending on the decision maker and naturally the models have difficult to capture these details.

In general, the model exceeded the expectations when achieved the outstanding results in terms of accuracy and f-score and proved reliable by keeping the same performance when facing different and shuffled datasets.

Reflection

The process used for this project can be summarized in the following steps:

1. An initial real-world problem and the related dataset were found
2. The data was download and preprocessed
3. A benchmark was created for the classifier
4. Multiple models were tested in order to identify the one that best suits the problem
5. Different parameters were tested in order to optimize the model
6. The best classifier was serialized and stored into a binary file

7. The application was designed to receive requests, load the pre-trained model and use it to make predictions

The most interesting aspect of the project is its applicability – the same structure can be useful for future projects or experiments like predicting the penalty applied to an employee that violate the rules, or even text classification.

Improvement

To achieve best results, the model should be able to receive feedback from its predictions, updating its data source from times to times.

Bob Carpenters' paper *Lazy Sparse Stochastic Gradient Descent for Regularized Multinomial Logistic Regression*[x] provides a valuable overview of logistic regression and describes online training for logistic regression by stochastic gradient descent under various parameters.

Although the dataset is pretty small, we can consider using 'sag' solver and 'warm_start' in the future in order to obtain faster convergence.

The user experience could also be improved by using text fields to identify categories of analysis or even make predictions based on the description of the facts. Indeed, it will become a new experience soon.

---

[i] https://github.com/udacity/machine-learning/tree/master/projects/finding_donors
[ii] http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html
[iii] https://en.wikipedia.org/wiki/Gradient_boosting
[iv] http://scikit-learn.org/stable/modules/ensemble.html#gradient-boosting
[v] https://en.wikipedia.org/wiki/Logistic_regression
[vi] http://classroom.synonym.com/disadvantages-logistic-regression-8574447.html
[vii] http://scikit-learn.org/stable/modules/ensemble.html#adaboost
[viii] https://en.wikipedia.org/wiki/AdaBoost
[ix] http://www.saedsayad.com/logistic_regression.htm
[x] https://lingpipe.files.wordpress.com/2008/04/lazysgdregression.pdf