Identify Fraud From Enron Email

Final Project

Student: Giulio Cesare Mastrocinque Santo

15/02/2019

Question 1: summarize for us the goal of this project and how machine learning is useful in trying to accomplish it. As part of your answer, give some background on the dataset and how it can be used to answer the project question. Were there any outliers in the data when you got it, and how did you handle those?

Enron Corporation was an American energy company based in Houston, Texas, formed in 1985 by **Kenneth Lay**. According to [1], when Enron's CEO **Jeffrey Skilling** ["was hired, he developed a staff of executives that - by the use of accounting loopholes, special purpose entities, and poor financial reporting - were able to hide billions of dollars in debt from failed deals and projects. Chief Financial Officer **Andrew Fastow** and other executives not only misled Enron's Board of Directors and Audit Committee on high-risk accounting practices, but also pressured Arthur Andersen to ignore the issues"].

Since a lot of information regardless Enron Corporation became public after the scandal, why not use our Data Scientist skills to understand what was going on? That is exactly the main purpose of these project: given a collection of financial and email data from Enron's employees, can we build a classifier to identify people of interest (POI), i.e., individuals that could be related somehow with the scandal? My guess is yes, we can.

Let's then first take a look at the data we have available:

Udacity kindly provide us a dictionary with the following characteristics:

- 146 individuals (data points)
- 18 people being market as interesting for the investigation
- 21 features per person

The features available can be seen bellow taking Jeffry Skilling as an example:

*{**'salary'**: 1111258, **'to_messages'**: 3627, **'deferral_payments'**: 'NaN', **'total_payments'**: 8682716, **'exercised_stock_options'**: 19250000, **'bonus'**: 5600000, **'restricted_stock'**: 6843672, **'shared_receipt_with_poi'**: 2042, **'restricted_stock_deferred'**: 'NaN', **'total_stock_value'**: 26093672, **'expenses'**: 29336, **'loan_advances'**: 'NaN', **'from_messages'**: 108, **'other'**: 22122, **'from_this_person_to_poi'**: 30, **'poi'**: True, **'director_fees'**: 'NaN', **'deferred_income'**: 'NaN', **'long_term_incentive'**: 1920000, **'email_address'**: 'jeff.skilling@enron.com', **'from_poi_to_this_person'**: 88}*

From the above example one could immediately notice an 'NaN' value! Let's take a look how many 'NaN' values we have for each feature considering all people:

{'salary': 51,

'to_messages': 60,

'deferral_payments': 107,

'total_payments': 21,

'loan_advances': 142,

'bonus': 64,

'email_address': 35,

'restricted_stock_deferred': 128,

'total_stock_value': 20,

'shared_receipt_with_poi': 60,

'long_term_incentive': 80,

'exercised_stock_options': 44,

'from_messages': 60,

'other': 53,

'from_poi_to_this_person': 60,

'from_this_person_to_poi': 60,

'poi': 0,

'deferred_income': 97,

'expenses': 51,

'restricted_stock': 36,

'director_fees': 129}

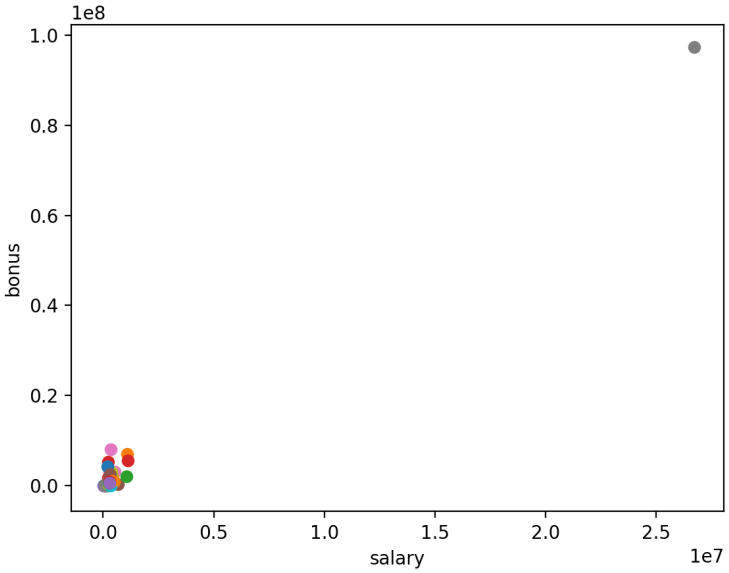
Good news is that we do not have missing values for the attribute POI! However, we have lots of missing values that must be treated. In order to find potential **outliers**, I filtered every person that had **more than 16 attributes with missing values (NaN)**. Then, I manually verified each of them and selected as outliers the following:

- People with almost no features at all: [WODRASKA JOHN, WHALEY DAVID A, CLINE KENNETH W, WROBEL BRUCE, SCRIMSHAW MATTHEW, GILLIS JOHN]
- An interesting "person" that appeared: THE TRAVEL AGENCY IN THE PARK

Finally, let's search for **too high** values, because they can make an algorithm wrongly points towards a POI. Printing out the highest **salary and bonus**, one obtains:

- Max Salary: 26704229.0
- Max Bonus: 97343619.0

Let's print out a scatter plot between those two features (Reference: Outliers Lesson):



If we search for the person with that feature values, we end up finding a person called **'Total'**, which was probably a transcription mistake. Removing the 'Total' element and recalculating these values, one obtains that the highest values now are associated with Jeffry Skilling and John J. Lavorato, both closely related to Enron's fraud. Therefore, those are probably not outliers.

With that being said, the following Keys were considered outliers and will be further removed from the dataset:

- [WODRASKA JOHN, WHALEY DAVID A, CLINE KENNETH W, WROBEL BRUCE, SCRIMSHAW MATTHEW, GILLIS JOHN, THE TRAVEL AGENCY IN THE PARK, TOTAL]

Outlier Removal

To treat the outliers, I first removed the above keys of the dataset. Then, the "NaN" values were handled. Several approaches could be used to replace the "NaN" values with some numeric one. For example, one could fill the salaries with an overall average. What I have done was to replace them with a 0 value for the numeric cases and with an empty string for the features that are strings (example: email address).

Question 2: what features did you end up using in your POI identifies, and what selection process did you use to pick them? Did you have to do any scaling? Why or why not? As part of the assignment, you should attempt to engineer your own feature that does not come ready-made in the dataset – explain what feature you tried to make, and the rationale behind it. (You do not necessarily have to use it in the final analysis, only engineer and test it). In your feature selection step, if you used an algorithm like a decision tree, please also give the feature importance's of the features that you use, and if used an automated feature selection function like SelectKBest, please report the feature scores and reasons your choice of parameter values.

The first thing I did was to follow the approach of lesson 12: Feature Selection. In this lesson we studied the *from_poi_to_this_person* and the *from_this_person_to_poi* features. These are good characteristics that could help identify people of interest having only a few of them in the dataset (imbalanced). It is a common sense that if someone trade too much emails with a POI maybe this person is itself a POI. What we also concluded in that lesson was that a more interesting measure is the proportion of emails sent/received to/from a POI amongst all the emails of that person. That is because this is a more relative measure. Imagine someone sent (or received) 100 emails to (from) a POI. That number only tell us something if it is compared to the number of emails that person sent in the same period: if the person sent 10000 emails, those 100 emails seems irrelevant, which would not be true in the case the person sent 150 emails.

Therefore, two features were added to the original 21: *fraction_to_poi* and *fraction_from_poi*. As an example, let's take Jeffry Skilling again:

> *{'to_messages': 3627, 'deferral_payments': 0, 'expenses': 29336, 'poi': True, 'deferred_income': 0, 'email_address': 'jeff.skilling@enron.com', 'long_term_incentive': 1920000,* ***'fraction_from_poi'****: 0.0242624758753791, 'restricted_stock_deferred': 0, 'shared_receipt_with_poi': 2042, 'loan_advances': 0, 'from_messages': 108, 'other': 22122, 'director_fees': 0, 'bonus': 5600000, 'total_stock_value': 26093672, 'from_poi_to_this_person': 88, 'from_this_person_to_poi': 30, 'restricted_stock': 6843672, 'salary': 1111258, 'total_payments': 8682716,* ***'fraction_to_poi'****: 0.2777777777777778, 'exercised_stock_options': 19250000}*

Next, to select which features to use, I have performed an exhaustive feature selection. In order to do this, I first used SelectKBest, as suggested in the question, and took the first 10 most powerful features. Those features and their corresponding score can be seen in the table below:

Table 1. Top 10 SelectKBest Scores

Feature	Score from SelectKBest
exercised_stock_options	23.174239916979793
total_stock_value	22.52220391204718
bonus	19.188893999160484
salary	16.5752378476586
fraction_to_poi	15.021566736092426
deferred_income	10.637829454958055
long_term_incentive	9.02466557131846
restricted_stock	8.474434131490149
total_payments	8.171248823686968
shared_receipt_with_poi	7.653215658588216

Then, with these features, I performed all the possible combinations [2] between them **that have at least 4 parameters** (to avoid some signature parameter that can introduce bias), which are: $\binom{10}{4}$, $\binom{10}{5}$, $\binom{10}{6}$, … , $\binom{10}{10}$. Then, I performed the following validation:

- For each parameter combination, I divided the data into training and testing sets with a StratifiedShuffleSplit.
- Three models were fitted **without tuning** (tuning will be performed later): Decision Tree Classifier, Gaussian Naïve Bayes, K Neighbors (MinMaxScaler was used in this case).
- The mean Recall and Precision were calculated for each model and for each feature combination.
- **All the scores were added, creating a total score.**
- The set of features with highest total score was chosen as the finalist.

Bellow two tables are shown: the first one contains the top-10 **total score** and its corresponding features set for the case where ***fraction_to_poi*** and ***fraction_from_poi*** were not considered. The second one displays the same results when these engineered parameters were considered.

Table 2. Top-10 Total Score Set of Features (no engineered features)

Total Score (sum of Recall and Prediction from each model)	Set of Features
2.2980357142857146	['exercised_stock_options', 'total_stock_value', 'bonus', 'deferred_income']
2.2492261904761905	['exercised_stock_options', 'total_stock_value', 'bonus', 'deferred_income', 'shared_receipt_with_poi']
2.1250054112554113	['exercised_stock_options', 'total_stock_value', 'salary', 'deferred_income', 'restricted_stock']
2.0777655677655678	['exercised_stock_options', 'total_stock_value', 'bonus', 'salary', 'deferred_income']
2.053538961038961	['exercised_stock_options', 'total_stock_value', 'bonus', 'total_payments']
2.0506944444444444	['exercised_stock_options', 'bonus', 'deferred_income', 'shared_receipt_with_poi']
2.0358730158730163	['exercised_stock_options', 'total_stock_value', 'salary', 'deferred_income']
2.0342857142857143	['exercised_stock_options', 'total_stock_value', 'bonus', 'deferred_income', 'long_term_incentive', 'restricted_stock']
2.0301190476190474	['exercised_stock_options', 'total_stock_value', 'bonus', 'salary', 'deferred_income', 'restricted_stock']
1.9879116716616716	['exercised_stock_options', 'total_stock_value', 'bonus', 'deferred_income', 'long_term_incentive']

Table 3 Top-10 Total Score Set of Features (with engineered features)

Total Score (sum of Recall and Prediction from each model)	Set of Features
2.2759361471861475	['exercised_stock_options', 'total_stock_value', 'fraction_to_poi', 'deferred_income', 'long_term_incentive']
2.25093253968254	['exercised_stock_options', 'total_stock_value', 'bonus', 'deferred_income']
2.2502579365079365	['exercised_stock_options', 'total_stock_value', 'fraction_to_poi', 'deferred_income', 'shared_receipt_with_poi']
2.2374603174603176	['exercised_stock_options', 'total_stock_value', 'bonus', 'salary', 'fraction_to_poi', 'deferred_income', 'long_term_incentive']
2.234563492063492	['exercised_stock_options', 'total_stock_value', 'bonus', 'deferred_income', 'shared_receipt_with_poi']
2.2221428571428574	['total_stock_value', 'fraction_to_poi', 'deferred_income', 'shared_receipt_with_poi']
2.194891774891775	['exercised_stock_options', 'total_stock_value', 'bonus', 'fraction_to_poi', 'deferred_income']
2.1667334054834058	['exercised_stock_options', 'total_stock_value', 'bonus', 'fraction_to_poi', 'restricted_stock', 'shared_receipt_with_poi']
2.165238095238095	['exercised_stock_options', 'total_stock_value', 'bonus', 'fraction_to_poi', 'deferred_income', 'shared_receipt_with_poi']
2.1636309523809523	['exercised_stock_options', 'total_stock_value', 'fraction_to_poi', 'deferred_income', 'restricted_stock', 'shared_receipt_with_poi']

It is important to point out that *fraction_to_poi*, which was one of the features added, ended up being a strong one (the fifth in the list above)! However, using *from_poi_to_this_person* and *from_this_person_to_poi* to engineer new features can potentially create bias because of our low proportion of POI. In order to see this, I will be comparing the final when these features are and are not included.

Notice that **data must be scaled because they are in a variety of different ranges and order of magnitude**. Not every classifier needs a scaled data. Those who need it are the ones that relies on the Euclidean distance between data points (in this report: K-nearest and SVM). I will be using the *MinMaxScaler* from Lesson 10.

Finally, the selected features were:

- **Considering the engineered parameters:**
  ['exercised_stock_options', 'total_stock_value', 'fraction_to_poi', 'deferred_income', 'long_term_incentive']
- **Not considering the engineered parameters:**
  ['exercised_stock_options', 'total_stock_value', 'bonus', 'deferred_income']

Question 3: what algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms?

I compared the performance of four different supervised algorithms:

- A Support Vector Machine (SVM)
- A Binary Tree
- A K-nearest Neighbors
- A Gaussian Naive Bayes Classifier

All algorithms were tuned using StratifiedShuffleSplit (the same method used in the tester.py script) and *GridSearchCV* as explained in the next question. The one that resulted in the best performance was the **Gaussian Naïve Bayes Classifier**, which was the finalist.

The following tables summarizes the scores obtained after tuning for each algorithm in the cases where ***fraction_to_poi*** and ***fraction_from_poi*** are and are not considered:

- **Considering *fraction_to_poi* and *fraction_from_poi* in the analysis**
  - **K-Nearest Neighbors: n_neighbors = 3**
  - **Decision Tree: min_samples_split = 13**
  - **SVM: kernel = rbf, C =1000 , gamma = 2**

Algorithm	Recall	Precision	F1 Score	Accuracy
K-Nearest Neighbors	0.16600	0.51393	0.25094	0.85843
Decision Tree	0.25000	0.33807	0.28744	0.82293
SVM	0.08350	0.15378	0.10823	0.80343
GaussianNB	0.40200	0.49660	0.44432	0.85636

- **Not considering *fraction_to_poi* and *fraction_from_poi* in the analysis**
  - **K-Nearest Neighbors: n_neighbors = 1**
  - **Decision Tree: min_samples_split = 14**
  - **SVM: kernel = rbf, C = 1000, gamma = 1**

Algorithm	Recall	Precision	F1 Score	Accuracy
K-Nearest Neighbors	0.41783	0.33050	0.36907	0.83857
Decision Tree	0.36735	0.27000	0.31124	0.82929
SVM	0.09850	0.32998	0.15171	0.84264
GaussianNB	0.39550	0.50350	0.44301	0.85793

Notice: to use the tester.py script given in the course for the SVC, I had to use a pipeline in order to pass the scaler and the classifier together.

Question 4: what does it mean to tune the parameters of an algorithm, and what can happen if you don't do this well? How did you tune the parameters of your particular algorithm? What parameters did you tune? (Some algorithms do not have parameters that you need to tune – if this is the case for the one you picked, identify and briefly explain how you would have done it for the model that was not your final choice or a different model that does utilize parameter tuning, e.g., a decision tree classifier).

A very nice explanation about tuning is given in [3]. Essentially, when choosing a Machine Learning algorithm, depending on the algorithm that is being used, a lot of parameters must be selected. A model parameter determines some of its characteristics. For example, in an SVC algorithm, one could choose different types for its Kernel (which could be linear, polynomial, sigmoid, etc), different values for its penalty parameter "C", and also different values for the kernel coefficient "gamma". Usually, those parameters are involved in a trade-off between smooth decisions boundaries and highest precision of the algorithm. Therefore, the basic idea of tuning is to choose a set of parameters that optimize the bias-variance trade-off, i.e., that makes your algorithm the simplest possible and also with the best possible predictability and generality.

The tuning procedure that I adopted in this project was the following:

- Create the labels and features using *featureFormat* and *targetFeatureSplit*.
- Iterate over a **StratifiedShuffleSplit** (it is important to use a Stratified operation because our classes are imbalanced) using GridSearchCV (with cv = sss) for a collection of parameters. The best parameters were chosen to build the final classifier.

The collection of parameters used for each model was:

- KNeighbors: a variety of n_neighbors values were tested.
- Decision Tree: a variety of min_sample_split values were tested.
- SVC: different kernels and values of C and gamma were teste.
- Gaussian Naïve Bayes: no parameter for tuning.

Question 5: what is validation, and what's a classic mistake you can make if you do it wrong? How did you validate your analysis?

Validation is probably the most important step in Machine Learning, because all the previous steps are useless without some evaluation metric. Validation is the process where a **trained** model is evaluated with a **testing** data set [4].

The validation process can be done in different ways. For example, one could simply divide the whole data into its corresponding training and testing sets using train_test_split, fit a desired model using the training data and then use some metric to see how the model performed over the testing data. However, this procedure may not work every time, as was the case in this project.

Sometimes data can be imbalanced, which means we have much more examples relative to a particular class than to another. In this case, a simple splitting can make the result of your validation very unreliable. In this case, stratified splitting can be used before evaluating the model. Moreover, it is also possible to do what is called cross-validation. In this type of validation, different groups of training/testing data are performed. In this project, this approach was used in a stratified fashion. A very common time of cross-validation in the Exhaustive one.

Choosing different kinds of metric is also important to correctly understand the performance of your algorithm. Four types of metrics are explained in the next question.

When performing validation, one must be aware of some important points:

- A too good result can be an indicative that something was probably done wrong. Some problems that could explain a skewed result: overfitting, validating the data over the whole data (or over the training data), presence of some signature feature (i.e. some feature that is always present in the data and that allows perfect classification – in that case, maybe you don't need machine learning).
- Usually, a good result in one metric not necessarily means your algorithm is performing well (specially in imbalance data). Therefore, more than one metric must be evaluated.

Question 6: give at least 2 evaluation metrics and your average performance for each of them. Explain an interpretation of your metrics that says something human-understandable about your algorithm's performance.

Four metrics were used in this project and are displayed in the table of "Question 3". First of all, I will give a briefly explanation of each one:

- **Accuracy**: essentially, it says how many points your algorithm correctly classified over all points.
- **Recall**: is how well your algorithm is able to identify the desired class inside a group of classes. A very good recall implies a lower precision, because we are forcing our algorithm to recognize the desired class at the cost of allowing it to consider other classes as the desired one.
- **Precision**: is the proportion of correct classified points for a given class. If your algorithm is too precise, it means you can really trust in its predictions. On the other hand, it will not be able to predict a considerable amount of points (which would mean a high recall)
- **F1**: "It is the best of the worlds". A high value of f1 score means reliability and good score. It is a "combination" of the above metrics.

Now, let's take as example the final classifier, which is the Naïve Bayes one. The obtained metrics were:

- **Accuracy:** 0.85793. That means 85,793% of all points were correctly classified as POI (people of interest) **or** Non-POI.
- **Recall:** 0.39550. That means amongst POI and Non-POI in the testing set, 39,550% of the POI were identified.
- **Precision:** 0.50350. That means that taking **all the classified points**, 50,350% of them were correctly classified as POI.

REFERENCES

[1] https://en.wikipedia.org/wiki/Enron_scandal

[2] https://stackoverflow.com/questions/464864/how-to-get-all-possible-combinations-of-a-list-s-elements

[3] https://stackoverflow.com/questions/22903267/what-is-tuning-in-machine-learning

[4] https://link.springer.com/referenceworkentry/10.1007%2F978-1-4419-9863-7_233

[5] https://stackoverflow.com/questions/2793324/is-there-a-simple-way-to-delete-a-list-element-by-value

[6] https://stackoverflow.com/questions/21939652/insert-at-first-position-of-a-list-in-python

[7] https://machinelearningmastery.com/k-fold-cross-validation/

[8] https://stackoverflow.com/questions/3121979/how-to-sort-list-tuple-of-lists-tuples

[9] https://stats.stackexchange.com/questions/43943/which-search-range-for-determining-svm-optimal-c-and-gamma-parameters

[10] https://sebastianraschka.com/faq/docs/scale-training-test.html