Identify Fraud from Enron Email

Data Exploration

The goal for this project is to use data from the Enron dataset which contains financial and email information from people involved in the Enron scandal to build build a predictive model which could identify POIs (persons of interest).

Data Structure

The master db contains information of each person with the following structure:

- Financial data:
  - `bonus`
  - `deferral_payments`
  - `deferred_income`
  - `director_fees`
  - `email_address`
  - `exercised_stock_options`
  - `expenses`
  - `loan_advances`
  - `long_term_incentive`
  - `other`
  - `restricted_stock`
  - `restricted_stock_deferred`
  - `salary`
  - `total_payments`
  - `total_stock_value`
- Email data:
  - `from_messages`
  - `to_messages`
  - `from_poi_to_this_person`
  - `from_this_person_to_poi`
  - `shared_receipt_with_poi`
- POI
  - `poi`

Here are a few examples extracted from our output (`console-exec.log`):


```
{
  "METTS MARK": {
    "bonus": 600000,
    "deferral_payments": "NaN",
    "deferred_income": "NaN",
    "director_fees": "NaN",
    "email_address": "mark.metts@enron.com",
```

```
      "exercised_stock_options": "NaN",
      "expenses": 94299,
      "from_messages": 29,
      "from_poi_to_this_person": 38,
      "from_this_person_to_poi": 1,
      "loan_advances": "NaN",
      "long_term_incentive": "NaN",
      "other": 1740,
      "poi": false,
      "restricted_stock": 585062,
      "restricted_stock_deferred": "NaN",
      "salary": 365788,
      "shared_receipt_with_poi": 702,
      "to_messages": 807,
      "total_payments": 1061827,
      "total_stock_value": 585062
    }, "GLISAN JR BEN F": {
      "bonus": 600000,
      "deferral_payments": "NaN",
      "deferred_income": "NaN",
      "director_fees": "NaN",
      "email_address": "ben.glisan@enron.com",
      "exercised_stock_options": 384728,
      "expenses": 125978,
      "from_messages": 16,
      "from_poi_to_this_person": 52,
      "from_this_person_to_poi": 6,
      "loan_advances": "NaN",
      "long_term_incentive": 71023,
      "other": 200308,
      "poi": true,
      "restricted_stock": 393818,
      "restricted_stock_deferred": "NaN",
      "salary": 274975,
      "shared_receipt_with_poi": 874,
      "to_messages": 873,
      "total_payments": 1272284,
      "total_stock_value": 778546
    }
  }
```


There is a total of 146 "persons" (or datapoints) on the data set, our of which 18 are POIs.

The POIs are identified as followed: `HANNON KEVIN P, COLWELL WESLEY, RIEKER PAULA H, KOPPER MICHAEL J, SHELBY REX, DELAINEY DAVID W, LAY KENNETH L, BOWEN JR RAYMOND M, BELDEN TIMOTHY N, FASTOW ANDREW S, CALGER CHRISTOPHER F, RICE KENNETH D, SKILLING JEFFREY K, YEAGER F SCOTT, HIRKO JOSEPH, KOENIG MARK E, CAUSEY RICHARD A, GLISAN JR BEN F`

The data exploration features are described by code on the file `src/helpers/analyse.py`

Outliers

The outliers detection covers 4 steps:

1. Identify cases with lots of missing data

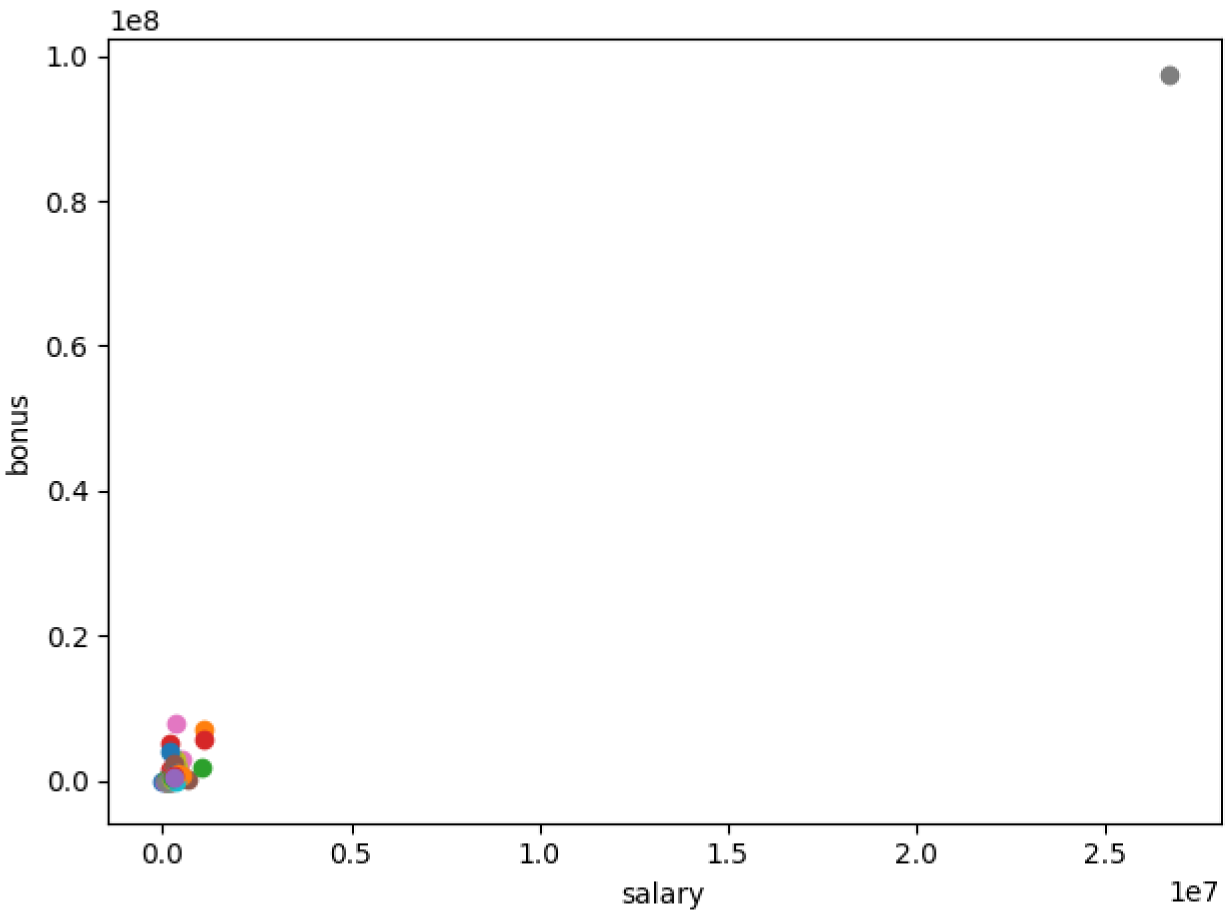
   This condition was based on a few fields being NaN or 0, and accounted for the following fields:

   - salary
   - bonus
   - total_payments
   - from_poi_to_this_person
   - from_this_person_to_poi
   - total_stock_value
   - from_messages
   - to_messages

   When a "person" meets all of this fields under the criteria, it is considered a potential outlier.

2. Identified salaries out of the normal

   This validation is performed by validating the `salary` and `bonus` fields for values out of the normal. For that we used the following plot:

   At simple sight we can see at least one value being out of proportions with the rest, so we filter them out, the condition was as follow:

   `data[person]['salary'] > 800000 or data[person]['bonus'] > 6000000`

3. Investigate the "persons" (keys) properly

   This is the result of manually looking at the cases, person by person and noticing that 1 of the "person" records is actually a company name: `THE TRAVEL AGENCY IN THE PARK`

4. Re-validate potential outliers

   Just to make sure we are not marking anyone critical as outlier we drop from the outlier list those who meet the following criteria:

   1. Are POIs
   2. Have high interactions with a POI

After the full analysis we were able to identify the following outliers:

Person	Reason
CHAN RONNIE	Incomplete Data
LOCKHART EUGENE E	Incomplete Data
TOTAL	Summary Row
THE TRAVEL AGENCY IN THE PARK	Not a Person

The outlier detection and extraction features are described by code on the file `src/helpers/outliers.py`

Feature selection

show the Initially we started off the analysis with all the features but this happened to carry a lot of unimportant or irrelevant fields for processing I applied an algorithm to detect relevant fields and a separate logic to add new features.

Find optimal features

The method `find_optimal_features` returns a list with the 10 most relevant features according to the algorithm `SelectKBest`. This filter is important to reduce the noise of data for further steps.

The method calculates and sorts the features according to the K highest scores. Here are the results for our optimal run:

Feature	Score	p-value
Salary	2.99	0.084
total_payments	2.75	0.097
loan_advances	6.63	0.010
bonus	5.05	0.025
total_stock_value	5.41	0.020
expenses	1.45	0.229

Feature	Score	p-value
exercised_stock_options	6.76	0.009
other	1.69	0.193
long_term_incentive	2.50	0.114
shared_receipt_with_poi	2.38	0.123

We will see later on this report the impact of running the classifiers before and after applying the feature selection.

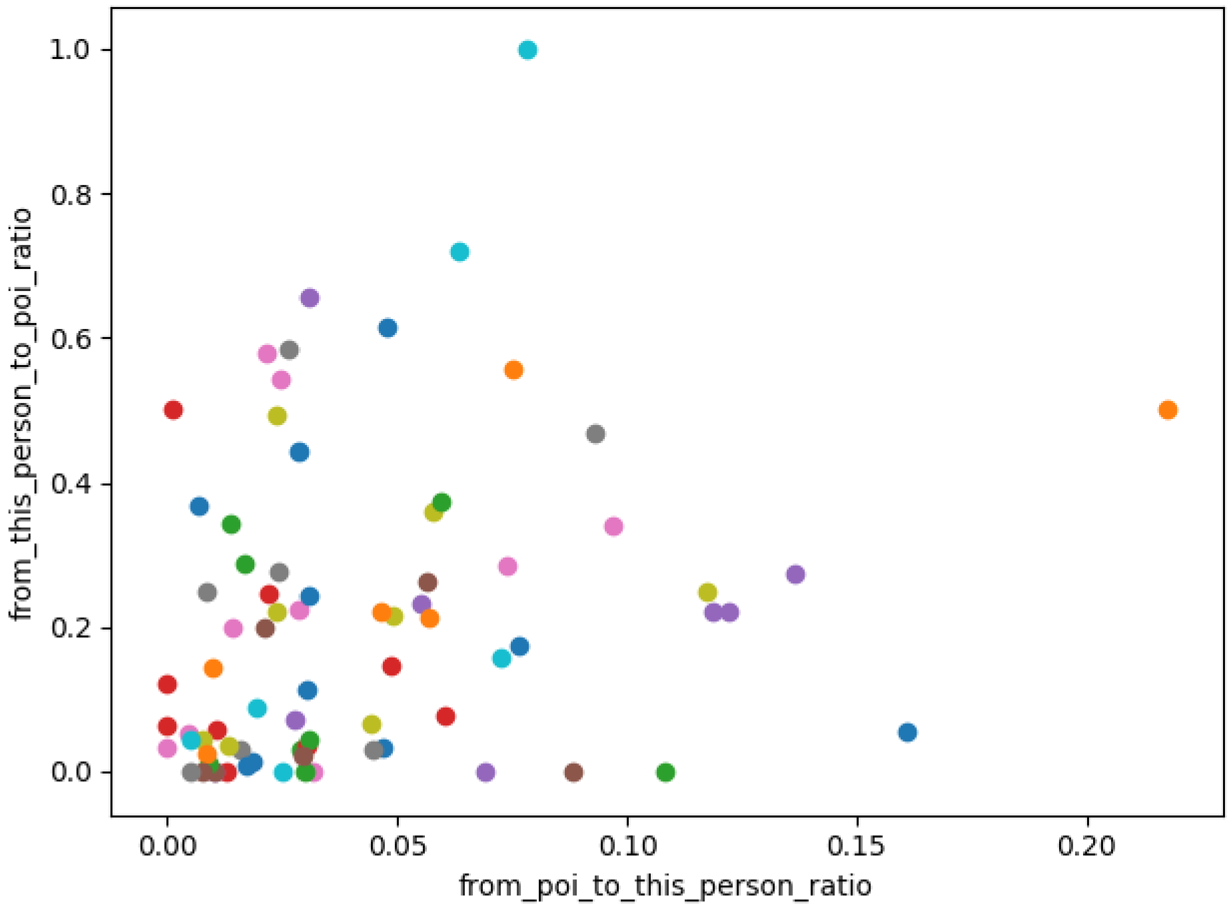
Adding new features

New features were created by processing rations of the information we had, here are the new fields:

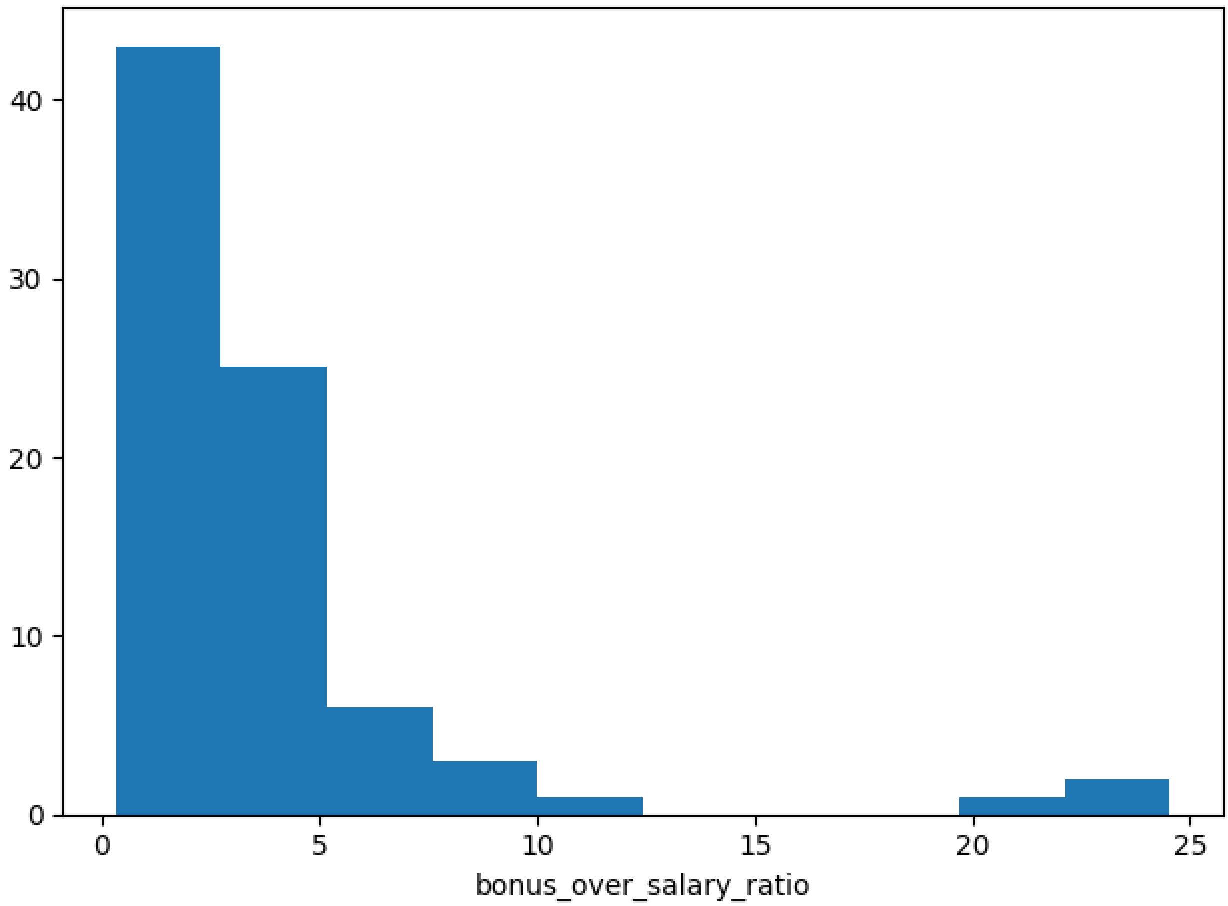
1. `from_poi_to_this_person_ratio`: messages from POIs / total messages received
2. `from_this_person_to_poi_ratio`: messages to POIs / total messages sent
3. `bonus_over_salary_ratio`: simply bonus / salary

The following charts detail this ratios:

Chatter plot highlighting the messages ratios



Histogram covering the bonus over salary ratio



Result

The final contains the following features:

- `poi`
- `salary`
- `total_payments`
- `loan_advances`
- `bonus`
- `total_stock_value`
- `expenses`
- `exercised_stock_options`
- `other`
- `long_term_incentive`
- `shared_receipt_with_poi`

The feature selection is described by code on the file `src/helpers/features.py`

Classifiers

In this section we will explore the different classifiers we tried and finally we are going to pick the one which gave us the optimal results.

Complete Feature List				**Optimal Feature List**		
Algorithm	**Accuracy**	**Precision**	**Recall**	**Accuracy**	**Precision**	**Recall**
Gaussian Naive Bayes	0.73573	0.21635	0.37450	0.79640	0.22263	0.21150
Ada Boost	0.82827	0.35185	0.34200	0.82127	0.33251	0.33251
Ada Boost (Tuned)	-	-	-	0.81280	0.32187	0.36500
Random Forest	0.85453	0.36458	0.12250	0.85987	0.42672	0.14850
Random Forest (Tuned)	-	-	-	0.85427	0.38902	0.22974
SVC	0.48747	0.14055	0.55600	0.47840	0.14138	0.57400
SVC (Tuned)	-	-	-	0.86320	0.12857	0.00450

- Precision: in simple words, precision tries to answer the question: `What proportion of positive identifications was actually correct?`
- Recall: in simple words, recall tries to answer the question: `What proportion of actual positives was identified correctly?`

Since the best results where achieved using Ada Boost, that's the one we selected for the final output.

Additionally is important to highlight the variation of the results by using different feature selection. Let's look at each algorithm separately

1. Gaussian Naive Bayes: Though increasing accuracy, we had a significant drop on recall
2. Ada Boost: Very interesting similar results with a slightly drop on recall
3. Random Forest: Slightly best results
4. SVC: Seems to have overall performed better with the complete feature list

We did not use scaling on the algorithms.

The feature selection is described by code on the file `src/helpers/analyse.py`

On Model Tuning

In order to provide a degree of parametrization to the algorithms we will provide support for algorithm tuning through `GridSearchCV`.

Every machine learning algorithm already supports hyper parameters that can be set and modified to deliver different results on the data.

Going through these different parameters, and trying out different values is important to to improve the performance over a particular dataset. Adjusting this parameters requires understanding each on of them, and applying unique values that matches your data. The same set of values can perform very different on different data, and thus is important to play and set different configurations, and find the optimal values for our case.

Since it can be overwhelming to try every possible combination to find the optimal results, we will use `GridSearchCV` module, as it was designed to automate this process.

Validation

Validation is the set of techniques to make sure the model performs well in a wide range of situations, and it's not just optimized for a particular data set or conditions.

Data validation is important to prevent for example, over-fitting.

This phenomena can be studied adjusting the amount of data points assigned to both training and testing sets. The most common way to test for it is with cross validation, a technique that dynamically assigns a percentage to the different sets.