ANALYSING THE ENRON DATASET

Intro to Machine Learning project by Andras Somi

1. Summarize for us the goal of this project and how machine learning is useful in trying to accomplish it.

Background

The goal of this project is to create a classifier to identify persons of interests in the infamous Enron scandal, based on some details on their remuneration and a dataset consisting many of their incoming and outgoing email messages.

Supervised machine learning is extremely well suited for this type of issues, as there are no obvious differences between 'POI' and 'non-POI' datapoints that would let us dissect the two groups manually, while the text data contains vast amout of emails that can only be processed in an automated fashion.

In this specific case we might also face some of the limitations, as the dataset consists of rather few persons (145 in total, of which only 18 is labelled as 'poi'). The small number of obsevations might make many solutions prone to overfitting, especially when using text-analysis techniques, which can produce vast number of additional features.

The data

- The dataset contains financial and email data about 145 persons related to the Enron scandal (146 data points in total, including an additional entry for financial totals). There are varying number of features for each individual.
- We have emailing data for 86 persons, all of them equally has all the five features present.

Number of observation for emailing features:

```
to_messages                86
shared_receipt_with_poi    86
from_this_person_to_poi    86
from_poi_to_this_person    86
from_messages              86
```

- There are varying number of observations in financial data. As this data comes from a seemingly complete financial report (`enron61702insiderpay.pdf`), we can assume that the missing values are actually zeros.

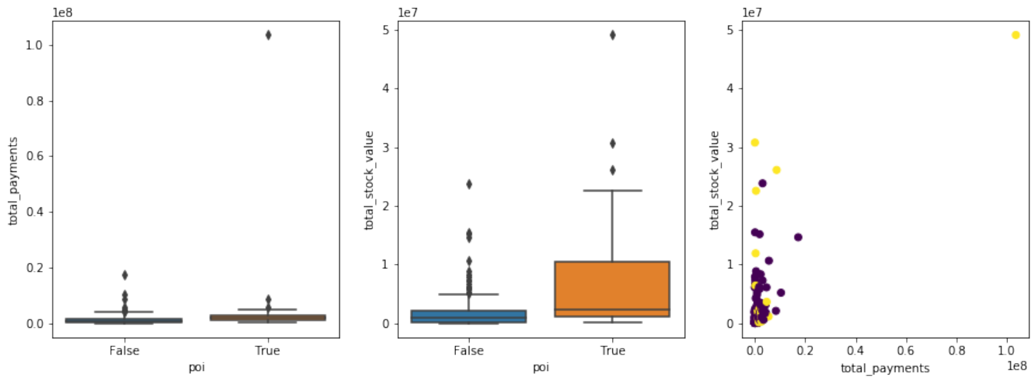
Number of observations for financial features

total_stock_value	125
total_payments	124
restricted_stock	109
exercised_stock_options	101
salary	94
expenses	94
other	92
bonus	81
long_term_incentive	65
deferred_income	48
deferral_payments	38
restricted_stock_deferred	17
director_fees	16
loan_advances	3

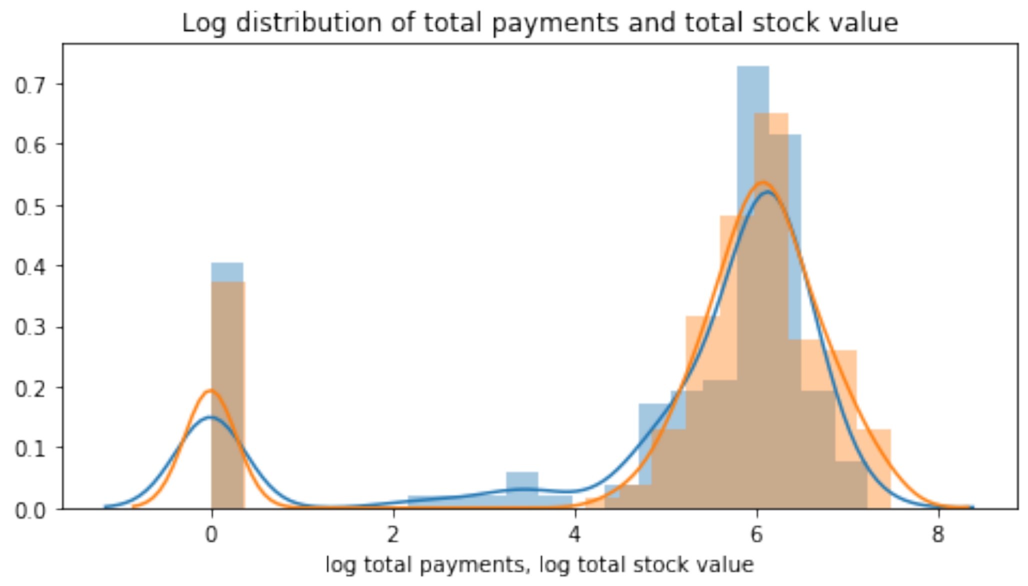
- On top of this we have email address for 111 persons and `poi` labels for all the people in the dataset.

Outliers

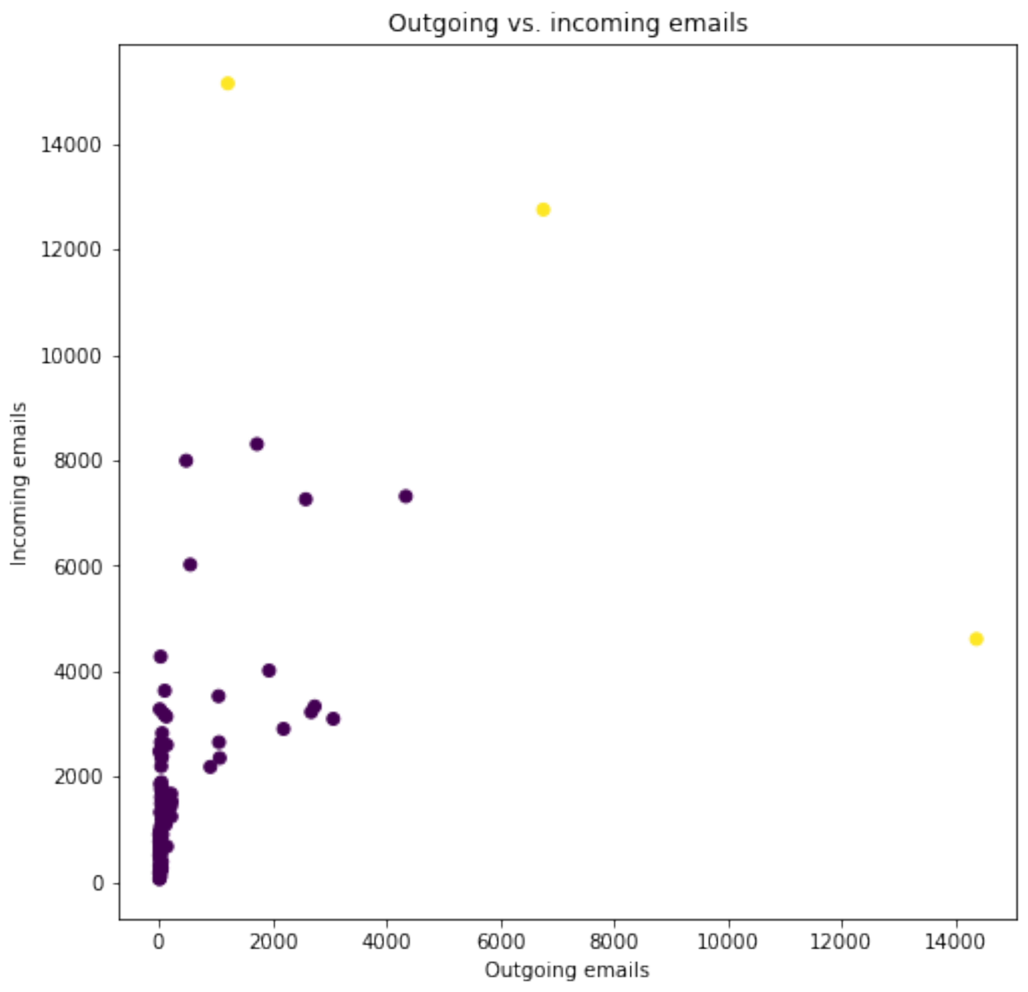
- There is an entry named 'TOTAL' for summing up the financial data. We have to exclude this from the analysis (this entry was already excluded for the observation counts above)
- The initial data dictionary contains errors in two persons' (Robert Belfer, Sanjay Bhatnagar) financial data, so the payments don't add up to the indicated totals. I manually corrected these entries based on the insider pay report.
- Kenneth Lay is an extreme outlier in total payments and stock value as well.



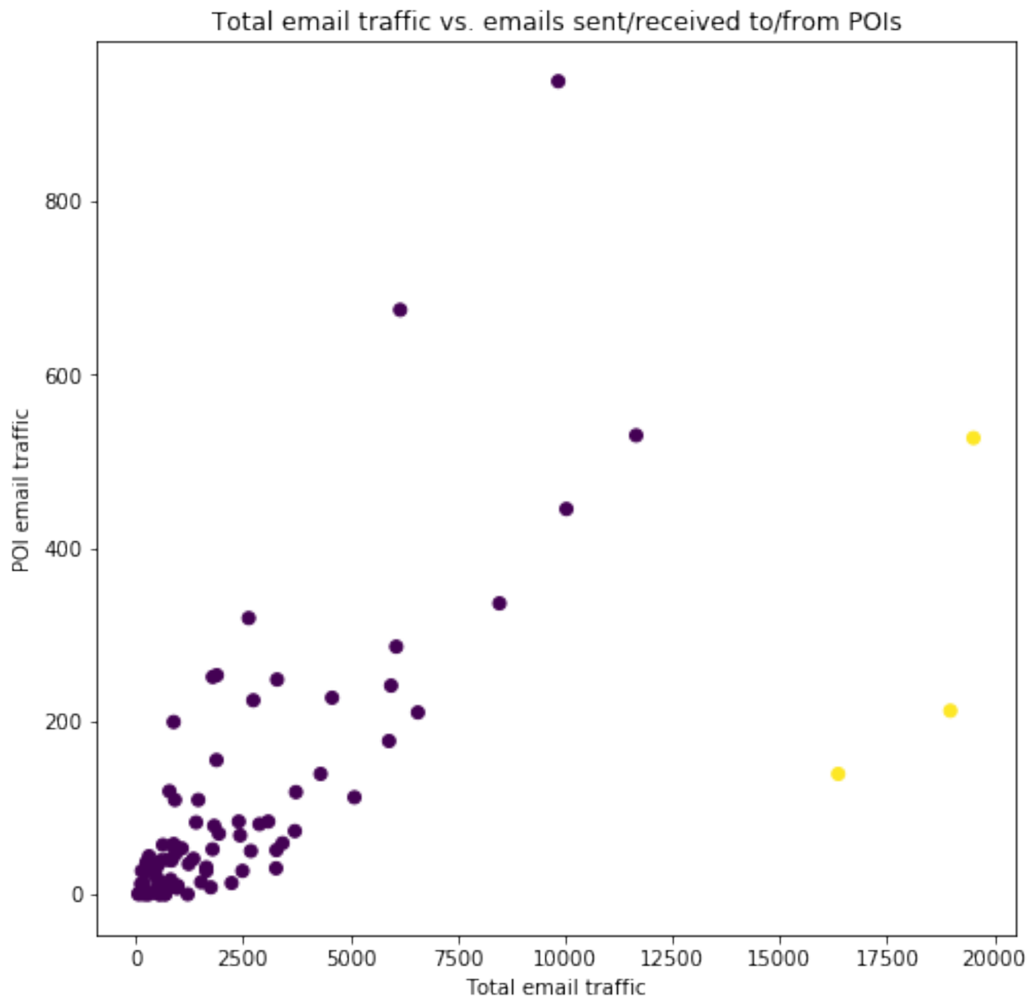
- Even keeping Ken Lay in the sample does not ruin the distribution of the two main financial features when plotted on log-scale. The distributions are slightly right-skewed, and show a secondary peak for the zero values.



- All the email features are heavily left-skewed, with some really extreme values. 3 persons have more than 10 thousand outgoing or incoming messages.



- I consider these outliers, especially when comparing the total email traffic (incoming + outgoing) to the email traffic with POIs (incoming + outgoing).



- One person (GLISAN JR BEN F) has a `shared_receipt_with_poi` value (874) larger than their overall number of incoming messages (873), which seems to be impossible (if I get these features right), so I exclude this as a data error.

2. What features did you end up using in your POI identifier, and what selection process did you use to pick them? Did you have to do any scaling? Why or why not?

Engineering features

I considered using the following constructed features:

Financial:

- **Total financial benefits:** The sum of total payments and total stock value. Shows the overall wealth received from Enron.
- **Adjusted payments**: Adding back the sum of deferred income, as these payments were already granted, but payable only in the future. This might reflect better the financial interest of the person in the operations of Enron.
- **Number of payment features:** How many different types of payments (bonus, salary, etc.) did a person receive from the company by counting the non-zero payment-related features. The bigger the score, the more complex the financial dependency of the person on Enron.
- **Ratio of payments to stock value:** This might also explain the nature of financial relationship with Enron.

Email:

- **Total email traffic:** Sum of sent and received messages. This can be an indicator of the overall activity of a person.
- **Poi email traffic:** Sum of sent and received messages to and from POIs. This can indicate how intensive was a person's colaboration with POIs.
- **Ratio of poi emails in in/outbox:** This might shed some light how intensive was the relationship between a given person and (other) POIs.

Text features:

I also tried to approach the problem from the raw emails. I created a processing and cleaning script for extracting the core of sent emails for every email address in the dataset (`process_emails.py`). I used the `Tf-idf vectorizer` of Scikit-learn to engineer features. Unfortunately I could not construct a classifier based on these text features that would pass the criteria of the assignment, so I left these out (it was fun anyway).

Selecting features
SelectKBest:

- I tried to select the best features by using `SelectKBest` and including all the default and engineered features.
- I scaled the features with `MinMaxScaler` to avoid issues later, as some of the financial variables are several order of magnitude higher than most of the email features. Some algorithm do not necessarily need scaling, but it does not do harm either, while others (eg. SVM) explicitly require it.
- In this scoring most of the email features got lower scores, and from the engineered features `total_benefits`, `payment_score` and `outbox_poi_ratio`

got to the top group.

- The first two features (`total_stock_value` and `total_benefits`) are strongly related, therefore I am not sure it would be wise to use both in a model.
- Skipping `total_benefits` I kept the other top 8 features (including `shared_receipt_with_poi`), as these show p-values that I consider sufficiently low (p < 0.01), meaning that in a univariate setup these proved to be statistically significant. In the later parts I excluded the remainder of the initial features.

Results of SelectKBest scoring using `f-classif` (feature, f-score, p-value)

```
p < 0.001
('total_stock_value', 16.071788624150162, 9.9718589223172257e-05),
('total_benefits', 15.732343770341879, 0.00011709074533943896),
('exercised_stock_options', 15.711758077612419,
0.00011823871082523517),
('bonus', 14.539408406389541, 0.00020676780046650225),
('deferred_income', 12.639742904307109, 0.00051889763355474007),
('salary', 12.588647839833962, 0.00053204003499508843),
('payments_score', 12.002000059238199, 0.00070977361827368249),
('outbox_poi_ratio', 11.749382202256147, 0.00080405643983 43786),

0.01 > p > 0.001
('shared_receipt_with_poi', 9.6989969436243388,
0.0022455816865628135),

0.05> p > 0.01
('poi_email_traffic', 6.1372284896171303, 0.014453064409554465),
('adjusted_payments', 5.8686433686774366, 0.016719666780097453),
('long_term_incentive', 5.7767664205655764, 0.017577736498077 35),
('from_poi_to_this_person', 4.6973875509540148,
0.031937041580804272),
('from_this_person_to_poi', 4.4308030511020684,
0.03712173001379751),
('to_messages', 4.3065366067886579, 0.039836648839694053),

p> 0.05
('restricted_stock', 3.9094348698440671, 0.050023095818164075),
('expenses', 3.3399882631653455, 0.069791074402897571),
('total_payments', 3.111431151712154, 0.079973605188852423),
('total_email_traffic', 2.7718813679165168, 0.098218689971936751),
('inbox_poi_ratio', 2.4025184842101162, 0.12344710241113975),
('director_fees', 1.9284396539717861, 0.16718280408898292),
('deferral_payments', 0.1900501417470408, 0.6635610435905972),
('loan_advances', 0.17954213479311426, 0.67243153498157426),
('from_messages', 0.080122111663662093, 0.77755925044305485),
('restricted_stock_deferred', 0.065197752973343959,
0.79884459974974731),
('payment_to_stock_value_ratio', 0.025200978120142464,
0.87410106840108548),
('other', 0.0047310688341011342, 0.94526282733110101)
```

- I also iterated the number of features for `SelectKBest` from 2 through 8 with the final model setup (including a PCA step in the pipeline), and scoring every best estimator of `GridSearchCV` by recall. This showed that including all the top 8 features (excluding `total_benefits`) gives the best result in this setup.

Here's the output, in descending order by recall scores, trained on whole dataset:

```
BEST PARAMETERS FOR 8 BEST FEATURES:
Score: 0.536690647482
Parameters: {'tree__min_samples_split': 2,
'selector__n_components': 4, 'tree__min_samples_leaf': 3}
---------------

BEST PARAMETERS FOR 7 BEST FEATURES:
Score: 0.423980815348
Parameters: {'tree__min_samples_split': 6,
'selector__n_components': 7, 'tree__min_samples_leaf': 2}
---------------

BEST PARAMETERS FOR 4 BEST FEATURES:
Score: 0.377458033573
Parameters: {'tree__min_samples_split': 13,
'selector__n_components': 4, 'tree__min_samples_leaf': 4}
---------------

BEST PARAMETERS FOR 6 BEST FEATURES:
Score: 0.367625899281
Parameters: {'tree__min_samples_split': 2,
'selector__n_components': 3, 'tree__min_samples_leaf': 8}
```

```
---------------

BEST PARAMETERS FOR 2 BEST FEATURES:
Score: 0.321103117506
Parameters: {'tree__min_samples_split': 3,
'selector__n_components': 2, 'tree__min_samples_leaf': 2}
---------------

BEST PARAMETERS FOR 5 BEST FEATURES:
Score: 0.301438848921
Parameters: {'tree__min_samples_split': 13,
'selector__n_components': 5, 'tree__min_samples_leaf': 4}
---------------

BEST PARAMETERS FOR 3 BEST FEATURES:
Score: 0.198561151079
Parameters: {'tree__min_samples_split': 2,
'selector__n_components': 2, 'tree__min_samples_leaf': 2}
---------------
```

(I also ran a version of the code where I tested every values between 1 and the number of available features as k for `SelectKBest()`, but I couldn't find better recall scores as with the 8 selected features. The code runs very long, therefore I did not include it into the final submission).

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

I tried multiple algorithms, and experimented more with decision tree, Naive-Bayes and SVM. Out of this three the decision tree produced the best results when fitted on the whole dataset. But this is due to overfitting, which I intended to handle by tuning the parameters. SVC did not work in this setup.

```
[('dt', 1.0, 1.0), ('nb', 0.3125, 0.35714285714285715), ('svc',
0.0, 0.0)]
```

For the final algorithm I used the 8 features selected in the previous step, but added a pipeline with a `PCA` step to potentially enhance the information extraction. Therefore I did not want to further narrow down the features list to let PCA alongside GridSearch extract the optimal amount of information.

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?

Parameter tuning is the process of finding the best parameters for the chosen algorithm (if it has parameters to tune). By changing parameters we can find the right trade-off between bias and variance, to obtain a model that sufficiently fits to training data but still generalize well for new data. If the parameters are not chosen carefully the model can be overfitted (low bias, high variance).

In this case, after manually trying many combinations of parameters, I decided to search for the best parameters with the help of `GridSearchCV` with 'recall' as scoring.

I tuned the number of selected components from PCA (`n_components`) and `min_samples_split` and `min_samples_leaf` in the decision tree to avoid overfitting. The former controls the minimum number of samples needed for considering a split for a node, the latter controls the minimum number of samples needed in a leaf node, both intended to keep the model from creating too many fragmented nodes and unnecessarily deep structure.

An example output:

```
BEST PARAMETERS FOR DT GRIDSEARCH:
{'selector__n_components': 4,
 'tree__min_samples_leaf': 3,
 'tree__min_samples_split': 2}
```

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

Classic mistake would be to stop here with the model evaluation, assessing the final classifier algorithm on the same data that we trained the algorithm on. This could mask the overfitting of the estimator, which in turn produces extraordinary results on the training set, but fails on other data.

The most important validaton technique is cross-validation, by using only a part of the dataset to train the model (in this case the pipeline) and evaluating its performance on the remaining data. This can be enhanced by using folding techniques (especially with smaller samples, like in this case).

I used `StratifiedShuffleSplit`, in order to handle the class imbalance of the data. The labels distribution is far from even in the dataset, therefore a simple train-test split or a K-Fold cross-validation could produce misleading results if the share of POIs in train and test sets are substantially different. `StratifiedShuffleSplit` makes sure that the different train-test splits contain about the same percentage of positive labels.

Instead of the default setting of `StratifiedShuffleSplit` for the train-test split, I raised the `test_size` parameter to 0.2 as the default value would result in test samples of 14 observations, which might be too low.

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.

I used four metrics to evaluate the algorithm: average precision, recall and accuracy scores for the testing sets, and also print out confusion matrix for every fold.

An example output from my cross-validation:

```
RESULTS FOR STRATIFIED SHUFFLE SPLIT CV:
Average precision: 0.5
Average recall: 0.541666666667
Average accuracy: 0.919642857143
```

In this specific example above the scores can be interpreted as follows:

- **accuracy:** in 92% of the cases on average the algorithm predicts the right POI label for a person.
- **precision:** when the algorithm labels a person POI, on average in 50% of the cases it turns out to be true.
- **recall:** when a person is a POI in reality, the algorithm gives the POI label 54% of the cases on average.

In this specific case accuracy might be a bit misleading, as most of the datapoints are labeled 'not-POI' (class imbalance), and as the algorithm predicts mostly negative results, it naturally 'hits' the true negatives in a relatively large number of cases. In the meantime the number of true positives is relatively low, hence the lower precision and recall values.

REFERENCES

1. Text feature extraction: http://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction
2. Classification of text documents using sparse features: http://scikit-learn.org/stable/auto_examples/text/document_classification_20newsgroups.html
3. Pipeline: chaining estimators: http://scikit-learn.org/stable/modules/pipeline.html
4. Pipeline example: http://scikit-learn.org/stable/auto_examples/plot_digits_pipe.html#sphx-glr-auto-examples-plot-digits-pipe-py
5. Selecting dimensionality reduction with Pipeline and GridSearchCV: http://scikit-learn.org/stable/auto_examples/plot_compare_reduction.html