Data Analyst nanodegree program

Intro to machine learning - Identify Fraud from Enron Email

Florina Georgescu

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? [RELEVANT RUBRIC ITEMS: “DATA EXPLORATION”, “OUTLIER INVESTIGATION”]

In this project, I use my new skills to build a person of interest identifier based on financial and email data made public as a result of the Enron scandal (one of the largest companies in the United States).The data contains a hand-generated list of persons of interest (POI) in the fraud case, which means individuals who were indicted, reached a settlement or plea deal with the government, or testified in exchange for prosecution immunity.

The data consists on 146 lines (the name of the employees) and 21 columns (features). 18 out of 146 are POI.

The features in the data fall into three major types, namely financial features, email features and POI labels.

```
financial features: ['salary', 'deferral_payments', 'total_payments',
'loan_advances', 'bonus', 'restricted_stock_deferred',
'deferred_income', 'total_stock_value', 'expenses',
'exercised_stock_options', 'other', 'long_term_incentive',
'restricted_stock', 'director_fees'] (all units are in US dollars)

email features: ['to_messages', 'email_address',
'from_poi_to_this_person', 'from_messages', 'from_this_person_to_poi',
'shared_receipt_with_poi'] (units are generally number of emails
messages; notable exception is ‘email_address’, which is a text
string)

POI label: [‘poi’] (boolean, represented as integer)
```

Data set information: Not all columns and rows have values. If a field is empty you can find it marked with 'NaN' POI can only have a boolean value (True =1, False =0)

1.1 MY RAW DATA SET AND FEATURES:

```
Data rows 146 dtypes: bool(1), float64(19), object(1) Index: 146
entries, METTS MARK to GLISAN JR BEN F Data columns (total 21
columns):
```

Column title	non-null data	data type
poi	146	bool
total_stock_value	126	float64
total_payments	125	float64
email_address	111	object

restricted_stock	110	float64
exercised_stock_options	102	float64
salary	95	float64
expenses	95	float64
other	93	float64
to_messages	86	float64
shared_receipt_with_poi	86	float64
from_messages	86	float64
from_this_person_to_poi	86	float64
from_poi_to_this_person	86	float64
bonus	82	float64
long_term_incentive	66	float64
deferred_income	49	float64
deferral_payments	39	float64
restricted_stock_deferred	18	float64
director_fees	17	float64
loan_advances	4	float64

In order to manipulate the data I have transformed the data set to pandas

The first step was to look for outliers as it can alter the final result. For that I have:

1.2 A. CHECKED OUT THE ROWS AS FOLLOWS:

Examined columns manually by printing the columns names and checked that the rows correspond to a name and surname ['TOTAL', 'THE TRAVEL AGENCY IN THE PARK'] where obviously data included by mistake Calculate percentage missing for each row and eliminate those persons that have very few data: LOCKHART EUGENE E has a 95.238095% of data missing

Therefore the outliers in terms of rows are:

```
outliers = ['TOTAL', 'THE TRAVEL AGENCY IN THE PARK', 'LOCKHART EUGENE
E']
```

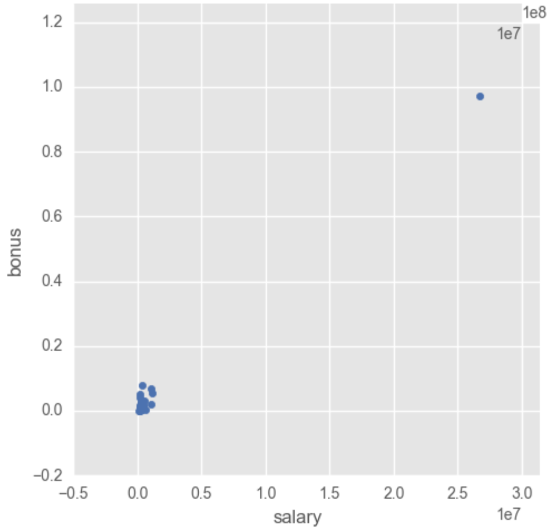
I also eliminated rows that do not have at least 70% of data available (max_data_percent)

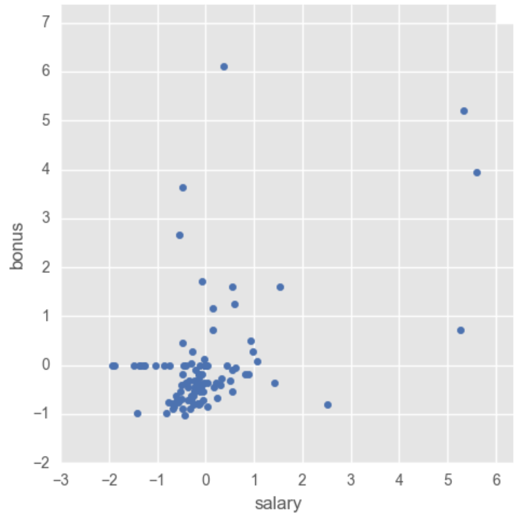
B. checked out the COLUMNS as follows

I did not consider of importance the e-mail (thus is not a value to be analysed and the 'from_poi_to_this_person', 'from_this_person_to_poi' as I later on used a combined feature of the two. 'deferral_payments', 'restricted_stock_deferred', 'loan_advances', 'director_fees' have less than 40 data points (27%) therefore I have removed this columns.

```
columns_to_delete = ['deferral_payments', 'restricted_stock_deferred',
'loan_advances', 'director_fees', 'from_poi_to_this_person',
'from_this_person_to_poi', 'email_address']
```

I have tried to visualize outliers but you have to have an intuitive view of what features you want to group. The plot below shows the trial (before and after eliminating outliers and normalizing.





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? 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 IMPORTANCES OF THE FEATURES THAT YOU USE, AND IF YOU USED AN AUTOMATED FEATURE SELECTION FUNCTION LIKE SELECTKBEST, PLEASE REPORT THE FEATURE SCORES AND REASONS FOR YOUR CHOICE OF PARAMETER VALUES. [RELEVANT RUBRIC ITEMS: “CREATE NEW FEATURES”, “PROPERLY SCALE FEATURES”, “INTELLIGENTLY SELECT FEATURE”]

At the beginning I have engineered a new column (feature) 'from_to_poi' that is the total of direct communication with the poi as = 'from_poi_to_this_person' + 'from_this_person_to_poi'. I have on purpose 'excluded shared_receipt_with_poi' because in my opinion people tend to exaggerate with the number of person they put in copy of an e-mail.

For feature selection I have used SelectKBest and the result is presented in the figure below. The selection for training/testing is only the k highest scoring features.

The feature I engineered does not have a high ranking 2.68 compared with 26.96 that is the highest but is on the list of BEST FEATURES.

So far so good until I got to the part of testing the algorithm with **test.py** But I’ll tell you all about that when we’ll talk about testing the algorithm. **Therefore from now on I’ll talk about two stages of my investigation (before and after-final)**

What is really clear for me now is that is of vital importance what you decide to use as paramenters and how you engineer them. It is important always to scale data but from the beginning.
I have created 2 new features a part of the first one:

```
# fraction_to_poi = from_this_person_to_poi/to_messages
enrondf['fraction_to_poi'] =
enrondf['from_this_person_to_poi']*1.0/enrondf['from_messages']

# fraction_from_poi = from_poi_to_this_person/to_messages
enrondf['fraction_from_poi'] =
enrondf['from_poi_to_this_person']*1.0/enrondf['to_messages']
```

If we don’t engineer these 2 features we’ll just give importance to one person that has “from_messages” the same importance indistinctively of how may it has sent to the POI. On the other hand we would have given the same importance to approximatively 400 allthough they send 4000 e-mail and 7000 e-mails (so one of them sends half as may as the other). When we would have normalized (by column) we would not have treated this.

So my lesson learned is: take a good look at the data if not you’ll wrongly predict! (we’ll see how wrong when we do the algorithm test).

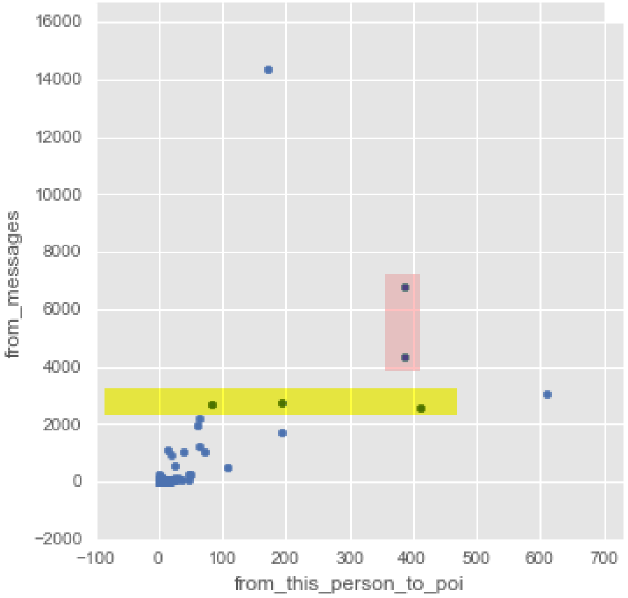


Figure 1 - From messages / from this person to POI

Score / BEST FEATURES (before)	**BEST FEATURES (after - final)**
26.96 exercised_stock_options	24.27 | exercised_stock_options
19.58 total_stock_value	17.29 | total_stock_value
10.76 bonus	11.54 | fraction_to_poi
8.84 salary	9.87 | bonus
7.05 total_payments	8.11 | salary
6.29 restricted_stock	5.96 | total_payments
5.61 long_term_incentive	5.42 | restricted_stock
5.39 shared_receipt_with_poi	5.14 | long_term_incentive
4.85 deferred_income	4.94 | shared_receipt_with_poi
2.68 from_to_poi	3.08 | deferred_income
1.76 other	2.46 | from_to_poi
0.55 from_messages	1.61 | other
0.55 expenses	1.00 | fraction_from_poi
0.35 to_messages	0.50 | from_messages
	0.50 | expenses
	0.32 | to_messages

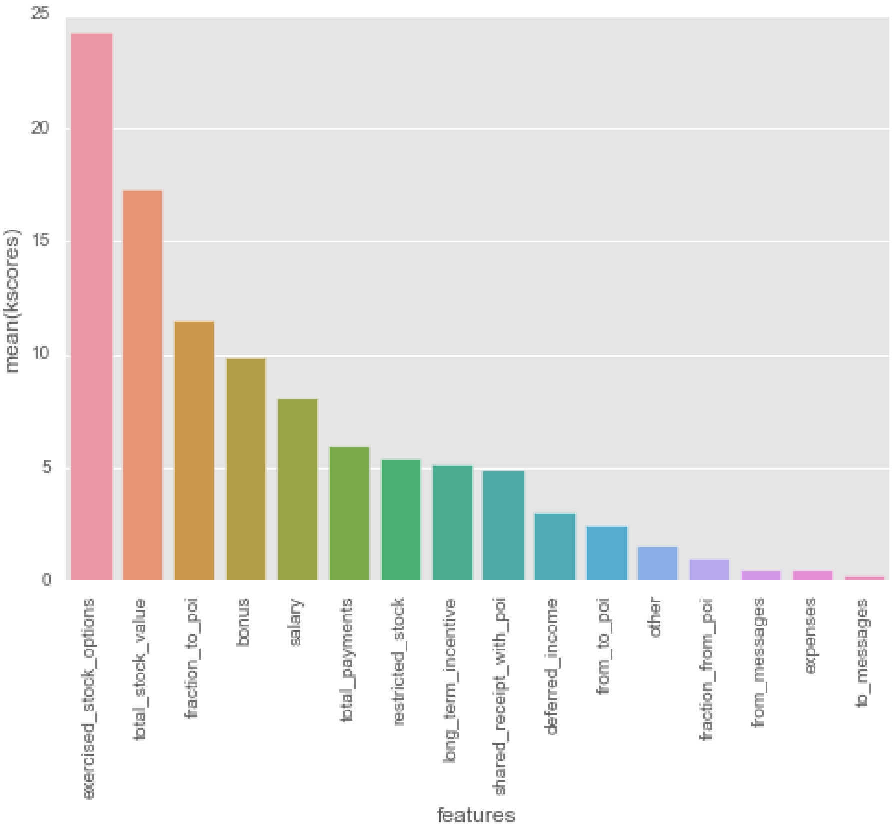


Figure 2 - Best Features - "after"

3. WHAT ALGORITHM DID YOU END UP USING? WHAT OTHER ONE(S) DID YOU TRY? HOW DID MODEL PERFORMANCE DIFFER BETWEEN ALGORITHMS? [RELEVANT RUBRIC ITEM: “PICK AN ALGORITHM”]

At the beginning I have only tested 2 algorithms

- Logistic Regression Classifier

```
Parameters:

'clf__C': [1e-08, 1e-07, 1e-06],
'clf__tol': [1e-2, 1e-3, 1e-4],
'clf__penalty': ['l1', 'l2'],
'clf__random_state': [42, 46, 60]

Parameters Tuned
    clf__C: 1e-08
    clf__penalty: 'l2'
    clf__random_state: 42
    clf__tol: 0.01
    feature_selection__k: 7
    features: exercised_stock_options    total_stock_value    bonus
salary    total_payments    restricted_stock    long_term_incentive
Result tuning: precision recall f1-score support
NON-POI        0.89       0.94       0.92        36
    POI        0.33       0.20       0.25         5
avg / total 0.83 0.85 0.84 41
Performance 1-Run:
Precision = 0.333333 and Recall = 0.20000
Decision Tree Classifier

Parameters:
```

```
"clf__min_samples_leaf": [2, 6, 10, 12],
"clf__min_samples_split": [2, 6, 10, 12],
"clf__criterion": ["entropy", "gini"],
"clf__max_depth": [None, 5],
"clf__random_state": [42, 46, 60]}

Parameters Tuned:
clf__criterion: 'gini'
clf__max_depth: None
clf__min_samples_leaf: 12
clf__min_samples_split: 2
clf__random_state: 42
feature_selection__k: 7
features: exercised_stock_options    total_stock_value    bonus
salary    total_payments    restricted_stock    long_term_incentive

Result tunning: precision recall f1-score support
NON-POI       0.91      0.89      0.90        36
    POI       0.33      0.40      0.36         5
avg / total 0.84 0.83 0.84 41
Performance 1-Run: Precision = 0.333333 and Recall = 0.40000
```

Therefore it seemed that the Decision Tree Classifier was the one to choose due to better precision and recall. But when I tested with tester.py I was totally surprised none of the two above had a precision & recall greater than 0.3.
The first thing that came in my mind was that the tunig was wrong, so I've tried different data splits and other tuning parameters. Then I thought that the data was wrongly processed, so I've tried eliminating more or less data etc.

After a long time of searches on github and a lot of questioning I've finally understood the importance of feature creation and data processing, so I've created the two parameters above mentioned.
In the after – final state I've tested one more algorithm (just to see that I did not choose a wrong algorithm for my data). The result below shows clearly not to trust the first run that for the alghoritm that finally passed the tester.py Decision Tree the first run Precision & Recall where a big zero.

```
LogisticRegression()
lr_params = {'clf__C': [1e-08, 1e-07, 1e-06],
             'clf__tol': [1e-2, 1e-3, 1e-4],
             'clf__penalty': ['l1', 'l2'],
             'clf__random_state': [42, 46, 60]}
Parameters Tuned:
        clf__C: 1e-08
        clf__penalty: 'l1'
        clf__random_state: 42
        clf__tol: 0.01
        feature_selection__k: 7
Result tunning:
             precision    recall  f1-score   support

    NON-POI       0.84      1.00      0.91        31
        POI       0.00      0.00      0.00         6

avg / total       0.70      0.84      0.76        37

Performance 1-Run:
Precision = 0.000000 and Recall = 0.00000


DecisionTreeClassifier()
dt_params = {"clf__min_samples_leaf": [2, 6, 10, 12],
             "clf__min_samples_split": [2, 6, 10, 12],
             "clf__criterion": ["entropy", "gini"],
             "clf__max_depth": [None, 5],
             "clf__random_state": [42, 46, 60]}
```

```
Parameters Tuned:
        clf__criterion: 'entropy'
        clf__max_depth: None
        clf__min_samples_leaf: 10
        clf__min_samples_split: 2
        clf__random_state: 42
        feature_selection__k: 7
Result tuning:
             precision    recall  f1-score   support

    NON-POI       0.82      0.87      0.84        31
        POI       0.00      0.00      0.00         6

avg / total       0.69      0.73      0.71        37

Performance 1-Run:
Precision = 0.000000 and Recall = 0.00000

##Support Vector Machine Classifier
print 'TRYING SUPPORT VECTOR MACHINES'
svc_clf = SVC()
svc_params = {'clf__C': [1000], 'clf__gamma': [0.001], 'clf__kernel':
['rbf']}

tune_and_eval_clf(svc_clf, svc_params, features, labels)

Parameters Tuned:
        clf__C: 1000
        clf__gamma: 0.001
        clf__kernel: 'rbf'
        feature_selection__k: 10
Result tunning:
             precision    recall  f1-score   support

    NON-POI       0.91      0.94      0.92        31
        POI       0.60      0.50      0.55         6

avg / total       0.86      0.86      0.86        37

Performance 1-Run:
Precision = 0.600000 and Recall = 0.50000

#### My final data set and features ####
Rows: 121
Columns:17
['poi', 'from_messages', 'to_messages', 'shared_receipt_with_poi',
'salary', 'bonus', 'total_payments', 'total_stock_value',
'deferred_income', 'exercised_stock_options', 'expenses',
'long_term_incentive', 'restricted_stock', 'other', 'from_to_poi',
'fraction_to_poi', 'fraction_from_poi']
```

3.1 RESULT WITH TESTER.PY

```
DecisionTreeClassifier(class_weight=None, criterion='gini',
max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=2,
            min_samples_split=6, min_weight_fraction_leaf=0.0,
            presort=False, random_state=42, splitter='best')
        Accuracy: 0.79562       Precision: 0.32647      Recall:
0.30900 F1: 0.31749     F2: 0.31234
        Total predictions: 13000        True positives:  618    False
positives: 1275   False negatives: 1382   True negatives: 9725
```

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? (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). [RELEVANT RUBRIC ITEM: “TUNE THE ALGORITHM”]

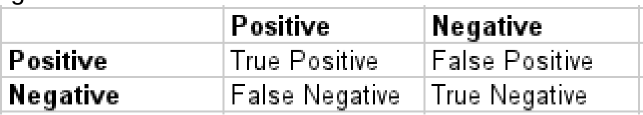
I have used Scikit learn - Grid Search - The traditional way of performing hyperparameter optimization (alghoritm tuning for machine learning) that is a grid search, or a parameter sweep, which is simply an exhaustive searching through a manually specified subset of the hyperparameter space of a learning algorithm. A grid search algorithm must be guided by some performance metric, typically measured by cross-validation on the training set] or evaluation on a held-out validation set. (source:https://en.wikipedia.org/wiki/Hyperparameter_optimization)
Tuning will influence the result. If not done well, the final model could be biased or over fitted and with low performance metrics.
We can observe that in the difference of the result if we maintain a fixed or random state. If a fix state is maintained we’ll always get the same result in the 1-Run and even more, when using the tester.py (with a larger data set 1000- Runs) the result changes again.

5. WHAT IS VALIDATION, AND WHAT’S A CLASSIC MISTAKE YOU CAN MAKE IF YOU DO IT WRONG? HOW DID YOU VALIDATE YOUR ANALYSIS? [RELEVANT RUBRIC ITEM: “VALIDATION STRATEGY”]

I do not know if my mistake was classic but at first I was validating with the first run, then I validated with my tuning and with only a small data set and then finally I used the tester.py (larger data set and 1000 runs) and finally got the good answer.
The goal of the cross validation is to define a dataset for testing the model by splitting a sample of data into subsets, performing the analysis on one subset (training set), and validating the analysis on the other subset (validation set/testing set).

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. [RELEVANT RUBRIC ITEM: “USAGE OF EVALUATION METRICS”]

Accuracy is the number of correct predictions made divided by the total number of predictions made.
The confusion matrix gives us more details about the next two metrics



	Positive	**Negative**
Positive	True Positive	False Positive
Negative	False Negative	True Negative

Precision is the number of True Positives divided by the number of True Positives and False Positives. Put another way, it is the number of positive predictions divided by the total number of positive class values predicted. It is also called the Positive Predictive Value (PPV).
Recall is the number of True Positives divided by the number of True Positives and the number of False Negatives. Put another way it is the number of positive predictions divided by the number of positive class values in the test data. It is also called Sensitivity or the True Positive Rate.

I used the Precision and Recall as in the project we here requested to have them both at least 0.3 but also because for me the Recall is the most important. We want to identify correctly the POIs.

7. REFERENCES:

Udacity nanodegree – Project P5 Intro to Machine Learning. videos & code

http://machinelearningmastery.com/how-to-tune-algorithm-parameters-with-scikit-learn/

https://www.civisanalytics.com/blog/workflows-in-python-using-pipeline-and-gridsearchcv-for-more-compact-and-comprehensive-code/

http://machinelearningmastery.com/classification-accuracy-is-not-enough-more-performance-measures-you-can-use/

https://github.com/petetheat/DAND_Udacity/blob/05c3cc54b3ff61b3a69e0cdc43fb4ca17e562498/P5/P5_Peter%20Eisenschmidt.pdf

https://github.com/seifip/udacity-data-analyst-nanodegree/tree/master/P5%20-%20Identifying%20Fraud%20from%20Enron%20Emails%20and%20Financial%20Data

8. FILES

poi_id.py : main file - POI identifier
my_dataset.pkl, my_classifier.pkl, my_feature_list.pkl – data generated
tester.py provided by Udacity
feature_format.py provided by Udacity