Bryony Miles - Enron Project Questions Answered 3 (more details in the report**)**

Question: Which Enron Employees may have committed fraud based on the public Enron financial and email dataset.

1. Data Exploration

146 entries, 21 features (financial, email related, **poi**)
poi = individuals who were indicted, reached a settlement or plea deal with the government, or testified in exchange for prosecution immunity

Machine learning is particularly useful in this case as the entries fall into two categories: guilty (poi) or not guilty (non-poi).

2. Outliers - during the data exploration I found the following:

- various NaN values which I analysed and converted to 0
- two invalid entries - TOTAL and THE TRAVEL AGENCY IN THE PARK which I deleted
- two data entry issues - BHANTNAGER SANJAY and BELFER ROBERT which I updated using the original data
- I also looked at two POIs with a lot of nulls and 20 non-POIs with very high entries but decided that the data was relevant and kept them in.
- the 2nd reviewer pointed out that non-POI 'LOCKHART EUGENE E' had all null values so I doubled checked this and deleted it too (cheating slightly)

3. New Features - In the end I creating 9 new features:

- *key_payments:* salary + bonus + other
- *deferral_balance:* deferral_payments + deferred_ income
- r*etention incentives:* long_term_incentive + total_stock_value
- *total_of_totals:* total_payments + total_stock_value
- *bonus/salary*
- *exercised_stock_options/salary*
- *retention_incentives/key_payments*
- *odd_payments - retention_incentives > 28,000,000* and *deferral_balance < -1,500,000*
- poi_emailratio - ratio of emails to or from POIs

4. **Feature Removal and Selection** - I missed out two features and then used KBest to score the remainder:

```
retention_incentives/key_payments KBEST 7.27958102488
bonus/salary KBEST 10.7835847082
poi_emailratio KBEST 5.39937028809
exercised_stock_options/salary KBEST 0.119303870047
odd_payments KBEST 47.4045185584
key_payments KBEST 17.4322739258
deferral_balance KBEST 13.5354825655
retention_incentives KBEST 23.8497535003
total_of_totals KBEST 16.9893364218
salary KBEST 18.2896840434
bonus KBEST 20.7922520472
other KBEST 4.20243630027
deferral_payments KBEST 0.228859619021
deferred_income KBEST 11.4248914854
loan_advances KBEST 7.18405565829
long_term_incentive KBEST 9.92218601319
director_fees KBEST 2.13148399246
expenses KBEST 5.41890018941
total_payments KBEST 9.28387361843
total_stock_value KBEST 22.5105490902
exercised_stock_options KBEST 22.3489754073
restricted_stock_deferred KBEST 0.768146344787
restricted_stock KBEST 8.82544221992
```

After considering GridSearchCV I went for the intuitive choice for now as I don't know which algorithm I'm going to use. The top 8 features all have a score above 17 and it tails off after that (14,11,11…).

5. Feature Scaling - the features are all integers, potentially related and lots of outliers of interest with varied scales I decided to use a MinMaxScaler. The second reviewer pointed out this does not affect the Decision Trees but this is still valid as I didn't know which algorithm I was going to choose.

6. Evaluation Metrics - I decided to use *precision* and *recall:*

- *Precision* = likelihood that an identified POI is actually a POI. % false alarms.
- *Recall* = likelihood of identifying a POI in the dataset if there was one.
- As the rubric asks for above 0.3 in both I'm assuming they are of equal importance.

I started with accuracy and found that the results were inappropriately high. Precision and recall take account of false positives and false negatives which are extremely important with this small, imbalanced dataset.

7. Cross Validation - I needed to split the data into training and testing to perform cross-validation. Why? To give an estimate on performance on an independent dataset and to double check that I wasn't overfitting. Initially I used train_test_split but then decided to take advantage of the validator in *tester.py* which uses Stratified Shuffle Split. This is recommended for a smaller imbalanced dataset to ward against small variations having too large an effect.

8. Algorithm Choice: I tested out five algorithms using the MinMaxScaler, KBest k=8. As advised after submission 1 I used a pipeline. The GaussianNB result was pretty good: *Precision* 0.47904, *Recall* 0.28 which is an indication I'm on the right track.

9. Parameter Tuning: I started out using GridSearchCV to tune the parameters but the search time was excruciatingly slow and I decided it would be a better learning experience if I iterated through the parameters myself and got a better idea of their properties. Using the Sklearn website, I worked systematically on the remaining four algorithms.

Why? Parameter tuning ensures you are getting the best performance from each algorithm. As you can see from my results parameter choices can have a significant positive and negative impact on initial results.

Decision Tree			**Logistic Regression**			**LinearSVC**			**Random Forest**		
	p	**r**		**p**	**r**		**p**	**r**		**p**	**r**
initial result	0.38872	0.37550		0.81074	0.15850		0.74405	0.25000		0.53076	0.22000
criterion = entropy	0.36359	0.33650	c = 1000	0.49851	0.25100	c=1000	0.29808	0.30300	n_estimators	improves balance not speed.	
splitter = random	0.40991	0.45500	As C increases, precision decreases and recall increases			loss='hinge'	1	0.27550	criterion = 'entropy'	0.52625	0.22050
max features	best at 7 (as you'd expect)		class weight = balanced	**0.32356**	**0.45800**	fit intercept = false	0.31150	0.24250	bootstrap ='false'	0.49271	0.30400
min_samples_split = 5	0.44651	0.37150	parameters with no impact	fit intercept = false, intercept scaling verbose, njobs		class weight = balanced	0.30794	0.48700	oob score n_estimators = 100	0.60670	0.27150
class weight = balanced	0.30546	0.26300				parameters with no impact	verbose, max_iter,dual		class weight = balanced	0.49195	0.16800
min_samples_split = 5 *and* splitter = random	**0.45385**	**0.35650**				I then looked at the various combinations and optimum result was.			As you can't have out of bag estimation if bootstrap=False and that had the best result I went for		
						C=10, class *and* weight = balanced	**0.31226**	**0.56050**	n_estimators = 100 *and* bootstrap = False	**0.50201**	**0.312**

10. Final Parameter Tuning

Finally I tested my best algorithm (Decision Tree, min_sample_splits = 5, splitter = random) with different KBest values:

Parameter	Precision	Recall
k=1	0.98909	0.27200
k=2	0.67005	0.29750
k=3	0.60310	0.33050
k=4	0.50850	0.32900
k=5	0.48873	0.33600
k=6	0.51581	0.41600
k=7	0.51617	0.42300
k=8	0.45385	0.3565
k=9	**0.48940**	**0.43850**
k=10	0.45008	0.42150
hard coded top 9 features	**0.52919**	**0.43050**
hard coded top 9 features + director_fees	0.48068	0.41050

The most balanced result was k = 9. I hard coded these features in using my KBest scores from earlier which produced a slightly better result. I then tested whether adding director_fees to the list would make a difference. The results were lower.

Therefore my final feature list is:

```
feature_list = ['poi','odd_payments','key_payments',
'retention_incentives','salary','bonus','total_stock_value',
'exercised_stock_options','total_of_totals','deferral_payments','director_fees']
```

and the results for Decision Tree, min_samples_split = 5, splitter = 'random' and random state = 42 (to maintain consistent results) are:

```
Bryonys-MBP:final_project bryonymiles$ python poi_id.py
Pipeline(steps=[('Scaler', MinMaxScaler(copy=True, feature_range=(0, 1))), ('SKB', SelectKBest(k=9,
score_func=<function f_classif at 0x107c63f28>)), ('Decision Tree',
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None, min_samples_leaf=1,
            min_samples_split=5, min_weight_fraction_leaf=0.0,
            presort=False, random_state=42, splitter='random'))])
      Accuracy: 0.87300 Precision: 0.52919     Recall: 0.43050   F1: 0.47477       F2: 0.44718
      Total predictions: 15000     True positives:  861    False positives:  766   False negatives:
1139  True negatives: 12234
```

11. Conclusion

So the final results means that 57% of POIs identified are definitely POIs and 47% of POIs in the dataset are being identified.

I'm sure there are other things I could do to improve this score. I didn't perform PCA and I did not use a Parameter Tuning algorithm. However, the major drawback is the restriction of a very small dataset (144 after outlier removal) with a small % of POIs - 12.5%. Ideally more data would be released for the 18 POIs identified who had no financial records in the dataset. This however is unlikely after so many years.

The logical next step therefore would be to look at the actual emails and tracking links between POIs and other email addresses and identifying key words which are more common when POIs are involved.

This has been a fabulous project. It has been a learning curve twice over and I would like to come back to the data again and have a dig into the email content linked to POIs.

The code is found in four python files:

- *explore_data.py*
- *feature_selection.py (commented out in this case)*
- *algorithms.py*
- *poi_id.py*