P5 Project: Identify Fraud from Enron Email

Questions

Kristofer Maanum

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”]

The goal of this project is to create a tool to identify potential 'persons of interest' ('POI') in the 2001 Enron scandal. To create this tool we have a database of Enron emails and an employee financial information. We have created an initial listing of POIs comprised of individuals who were indicted, reached a settlement or plea deal with the government, or testified in exchange for prosecution immunity. Using machine learning, we will be able to create a model or algorithm using the email and financial data as inputs and the POI labels as output.

The dataset includes Enron employees and their specific email and financial information. There are 146 observations in the dataset (people) and 20 variables (one of which is the POI label). The financial variables comprise two groups: stock information and direct payment information. Features “total_stock_value” and “total_payment” were sum totals of each of these groups, respectively, for each observation.

The main outlier or anomaly with the data is that there is a "TOTAL" line in the data which includes a sum of all email and financial features for the employees. This needed to be removed.

There are many areas where no data is given (“NaN” is listed). In all cases, this was taken to be a zero value. This appears to be consistent with actual data as we can use the “total_payment” and “total_stock_value” columns to work backwards and fill in the blanks. For example, for Mark Haedicke’s data below we can work out that ‘total_stock_value’ is the sum of ‘restricted_stock’, ‘restricted_stock_deferred’, and ‘exercised_stock_options’. Replacing the NaN with 0 for Kenneth Cline’s data is consistent with this. The same is true for the payment data.

Name	**restricted_stock**	**restricted_stock_ deferred**	**exercised_stock_ options**	**total_stock_ value**
CLINE KENNETH W	662086	-472568	NaN	189518
HAEDICKE MARK E	524169	-329825	608750	803094

We cannot work backwards to calculate this for the email data points. There are 59 persons with NaN for all email information. As such, email data may not be as useful to create the model. In further review these data proved to not be the best features to use in any case.

Some features have data points lying outside the mean, though in all cases these seem to be relevant and should be kept. The most outstanding is loan advances of over 80M USD to Ken Lay, but this feature wasn't used so the issue of whether this was an error is moot.

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"]

I ended up using the following:

- bonus
- deferred_income
- exercised_stock_options
- salary

In the selection process I first removed some features after EDA. Feature "email_address" was simply the person's email address which wouldn't give any useful information, so that was removed. Looking at the data closer, the financial data given was split into two groups: stock information and direct payment information. Features "total_stock_value" and "total_payment" were sum totals of each of these groups, respectively, for each observation. Since this information was already included in each individual data point I removed the totals.

For final feature selection I used SelectKBest with the following resulting scores:

Feature	**Group**	**SelectKBest Score**
exercised_stock_options	Stock	25.09754
bonus	Payment	21.06
salary	Payment	18.5757
deferred_income	Payment	11.59555
long_term_incentive	Payment	10.07245

restricted_stock	Stock	9.346701
shared_receipt_with_poi	E-mail	8.746486
loan_advances	Payment	7.24273
expenses	Payment	6.234201
from_poi_to_this_person	E-mail	5.344942
other	Payment	4.204971
pct_corr_with_POI	E-mail	3.735308
from_this_person_to_poi	E-mail	2.426508
director_fees	Payment	2.107656
to_messages	E-mail	1.698824
deferral_payments	Payment	0.217059
from_messages	E-mail	0.164165
restricted_stock_deferred	Stock	0.064984

To select the final features for use in the classifier I used GridSearchCV with different parameters. The top four features were selected.

I did not use feature scaling, even though it might usually be warranted for the k Nearest Neighbors classifier. Using standard scaling resulted in a worse result than when I did not use scaling. In considering this, I concluded that since all features included data in terms of U.S. dollars they are on the same "real world" scale, and the disparities between these values must provide better predictive power than scaling these disparities down.

I did create a new feature to see if the email information could add value. The feature I created was the percentage of a person's email that was to or from a POI ("pct_corr_with_POI"). The rationale is that it might make sense that persons colluding on fraud might have more correspondence with one another. This feature was created by the following equation:

$$(from\_poi\_to\_this\_person + from\_this\_person\_to\_poi) \div (from\_messages + to\_messages)$$

However, this feature did not add value to the model.

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"]

I ended up using a k nearest neighbors algorithm (KNeighborsClassifier). This returned the best performance by accuracy, precision and recall. I also tested Naive Bayes, a decision tree, and an SVM. The Naive Bayes algorithm came close to the performance of k nearest neighbors, but could not beat it. Decision tree and SVM were far behind.

The selection method was conducted using GridSearchCV with a number of different parameters. The best results from each are listed below (using the provided "tester.py" to generate results):

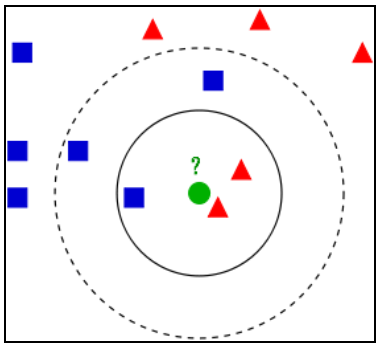
Algorithm/Model	Accuracy	Precision	Recall	F1
k Nearest Neighbors	0.860	0.514	0.390	0.443
Gaussian Naïve Bayes	0.857	0.501	0.341	0.405
Decision Tree	0.809	0.336	0.346	0.341
Support Vector Machine	0.848	0.427	0.196	0.268

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"]

Each algorithm has certain parameters that determine behaviour and outcomes of the algorithm. Tuning parameters means to run the algorithm with different parameters until you find the best parameters that return results that both make sense and are in line with your goals. For example, in the k Nearest Neighbors algorithm a new observation will be classified by a majority 'vote' of its neighbors. The observation will be assigned to the class most common among its *k* nearest neighbors. So, k is a parameter that might be tuned to get the best result.

In the below example the green observation is being classed by k Nearest Neighbors. By tuning k we might get the following two different outcomes:

- If k = 3, the nearest neighbors are the points within the solid circle. There are 2 red neighbors and 1 blue neighbor, so the new observation will be classed as "red"
- If k = 5, the nearest neighbors are the points within the dashed-line circle. There are 2 red neighbors and 3 blue neighbors, so the new observation will be classed as "blue"



Generally, parameters are tuned to increase algorithm accuracy. Within this understanding however, one might want to balance the algorithm toward a higher precision or recall. Practically, this means one can prioritize results exactness over completeness or vice versa.

When tuning the parameters it is also important to be careful about overfitting. Overfitting is when the algorithm is overly complex to maximize accuracy against a specific set of data. When the model is applied to a new set of data the model however it does not perform optimally. An overfit model maps to specific data points exactly whereas an optimal model will map to the data trend generally.

For this project I first used SelectKBest for feature selection then KNeighborsClassifier for the model. I used GridSearchCV to test a number of parameters and optimized for "recall" scoring. The reason for this was to select as many potential POIs as possible for further review, even if our precision was lowered a bit due to this. Key parameters used are the following:

- **SelectKBest**
  - *K*: 4
  - *Score function*: f_classif
- **KNeighborsClassifier**
  - *Algorithm*: 'brute'
  - *Metric / p*: Minkowski / 2 (Euclidean distance)
  - *N_neighbors ('k')*: 3
  - *Weights*: Distance

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"]

Validation is testing the performance of the model. This is done by testing the model against a set of data that is separate and different from the set of data used to train the data. The predicted results are then compared to actual labels to see how well the model performs.

If this is not done, or done incorrectly, the performance of the model might only be understood in relation to the training set of data alone. This can easily lead to overfitting as it would be impossible to tell whether a model maps to the specific data points of the training set or to the general trend of the data.

For my validation I used the a cross validation strategy of Stratified ShuffleSplit. This will create multiple iterations of train/test splits of data to check performance over each split.

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"]

My performance metrics for my model are as follows:

Algorithm/Model	Accuracy	Precision	Recall	F1
k Nearest Neighbors	0.860	0.514	0.390	0.443

Accuracy is an indication of how often the model is correct. So, in this case for the people included in the dataset the model was correct in identifying whether or not they were a POI.

Precision is how exact the model is in picking POIs. In this case, it is the percentage of predicted POIs are actually POIs.

Recall is how complete the model is in picking POIs. In this case it is the percentage of actual POIs that were flagged as such by our model.

In creating the algorithm I prioritized recall to “cast a wide net” and try to maximize the number of POIs identified, even if that meant that precision may have suffered slightly.