Identify Fraud from Enron Email

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 the project is to identify employees from Enron who may have committed fraud based on the public Enron financial and email dataset, i.e., a person of interest. We define a person of interest (POI) as an individual who was indicted, reached a settlement or plea deal with the government, or testified in exchange for prosecution immunity.

Machine learning algorithms are useful in trying to accomplish goals like this one because they can process datasets way faster than humans and they can spot relevant trends that humans would have a hard time realizing manually. Here is some background on the Enron financial and email dataset.

Employees

There are 146 Enron employees in the dataset. 18 of them are POIs.

Features

There are fourteen (14) financial features. All units are US dollars.

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

There are six (6) email features. All units are number of emails messages, except for ‘email_address’, which is a text string.

to_messages email_address from_poi_to_this_person from_messages from_this_person_to_poi shared_receipt_with_poi

There is one (1) other feature, which is a boolean, indicating whether or not the employee is a person of interest.

poi

Missing Features

20 of the 21 features have missing values (represented as "NaN"), with the exception being the "poi" feature.

The missing financial features are imputed by featureFormat to zero (0). Imputing to zero makes sense for these features because we have a reasonably complete financial picture through the FindLaw "Payments to Insiders" document. I am assuming that if a feature has a dash ('-'), like several 'bonus' rows do, that means it is zero. That isn't an unreasonable assumption since there are no actual zeros in that document and the dashes take their place.

I imputed the missing email features to each feature's mean. Imputing to zero doesn't make sense in this case because the email data appears incomplete. 60 of the 146 employees in the dataset

have "NaN" for all of their email features. A missing feature likely means we couldn't find the data, rather than the value is zero. Though this introduces some bias, we are at the whim of the dataset and imputing to the mean is a fine option.

Outliers

Before choosing the features to include in a machine learning learning algorithm, I plotted histograms of all of the features to get a feel for their distributions. These histograms made it easy to spot outliers, as well.

Every financial feature had a huge outlier generated by the "Total" row in the FindLaw "Payments to Insiders" document. I removed those from the dataset immediately. There was another non-employee entry in the dataset belonging to "The Travel Agency in the Park" that I removed as well.

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”]

Features Engineered

I add three features to the dataset to bring the total number of features to 24:

bonus_salary ratio from_this_person_to_poi_percentage from_poi_to_this_person_percentage Bonus salary ratio might be able to pick up on potential mischief involving employees with low salaries and high bonuses, or vice versa.

Scaling the **'from_this_person_to_poi' and 'from_poi_to_this_person'** by the total number of emails sent and received, respectively, might help us identify those have low amounts of email activity overall, but a high percentage of email activity with POIs.

Formula used is as below
POI_Interaction = from_this_person_to_poi+shared_receipt_with_poi+from_poi_to_this_person
Mail_Interaction = to_messages + from_messages fraction_POI_interaction = POI_interaction/Mail_interaction
This feature just gives a value if the person had a lot of mail communication with POIs which inturn could turn the spotlight on him/her and give us a clue as to whether he/she is a POI.

However it was observed that the fraction had values greater than 1 (which is because of data errors) and we are imputing the same to 1 just as an indication of maximum communication with pois

Scaling

Proprocessing via scaling was performed when I used the k-nearest neighbours algorithm and the support vector machines (SVM) algorithm, but not when I used a decision tree.

As described on stats.stackexchange (link for k-nearest neighbours; link for SVM), normalization is required for:

k-nearest neighbours because the distance between the points drives the clustering that determines the nearest neighbours. If one feature has a much larger scale than another, the clustering will be driven by the larger scale and the distance between points on the smaller scale will be overshadowed. SVM because the distance between the separating plane and the support vectors drives the algorithm's decision-making. If one feature has a much larger scale than another, it will dominate the other features distance-wise. Scaling isn't required for tree-based algorithms because the splitting of the data is based on a threshold value. This decision made based on this threshold value is not affected by different scales.

Selection Process

I used a univariate feature selection process, select k-best, in a pipeline with grid search to select the features. Select k-best removes all but the k highest scoring features. The number of features, 'k', was chosen through an exhaustive grid search driven by the 'f1' scoring estimator, intending to maximize precision and recall.

Features Selected

I used the following six features in my POI identifier, which was a decision tree classifier. The first number is feature importance (from the decision tree classifier) and the second is feature score (from select k-best). The order of features is descending based on feature importance.

To arrive at a constant selected featureset, multiple iterations of the above were performed and a union of the result along with the result yielded from Kbest was considered for the final list of 15 features.

Some intuition has also gone into the selection of the final list.Like the number of from_mails/to_mails cannot determine a person as a POI.The records having valid entries for restricted_stock_deferred or for deferral_payments are very less and hence can add negligible value to the algorithm.

Feature no. 1: bonus_salary_ratio (0.658595952706) (22.1067164085) Feature no. 2: shared_receipt_with_poi (0.180270198721) (6.1299573021) Feature no. 3: total_stock_value (0.161133848573) (16.8651432616) Feature no. 4: exercised_stock_options (0.0) (16.9328653375) Feature no. 5: bonus (0.0) (34.2129648303) Feature no. 6: salary (0.0) (17.7678544529) Bonus salary ratio, shared receipt with a POI, and total stock value were the most important features.

FEATURE SELECTION:

The existing features and the additionally engineered featured are fed into the ExtraTreesClassifier to get the feature_importances value for each of the feature and the same is used to trim the model to the relevant feature.

Choice of number of features:

A combination of Kbest and feature_importances from trees is used to arrive at the best features.

In the trees method adopted the median value of the importances are considered and those features with values greater than the median value is taken for further processing.The feature_ importances are dynamic and hence the number of features chosen also varies between 8-12.

To arrive at a constant selected featureset, multiple iterations of the above were performed and a union of the result along with the result yielded from Kbest was considered for the final list of 15 features.

Features	**KBest Scores**	**Feature_imp-iter 1**	**Feature_imp-iter 2**	**Feature_imp-iter 3**
Salary	7.57E+00	0.05855429	0	0.09498808
deferral_payments	6.92E-01	0	0	0
total_payments	5.13E+00	0.08315694	0.08315694	0.08315694
loan_advances	5.01E+00	0	0	0
bonus	1.12E+01	0.1113461	0	0.06580387
restricted_stock_deferred	1.06E-01	0	0	0
deferred_income	6.66E+00	0	0	0

PERFORMANCE Achieved with different combinations (Precision and Recall):

The decision tree classifier had a precision of 0.31336 and a recall of 0.59100, both above the 0.3 threshold. The SVM classifier (with features scaled) had a gaudy precision of 0.83333, but a poor recall of 0.06500. The k-nearest neighbours classifier had a precision of 0.32247 and a recall of 0.29200.

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 focused on three algorithms, with parameter tuning incorporated into algorithm selection (i.e. parameters tuned for more than one algorithm, and best algorithm-tune combination selected for final analysis). These algorithms were:

decision tree classifier SVM k-nearest neighbors Though I used the classfiication report for quick checks, I used tester.py's evaluation metrics to make sure I would get precision and recall above 0.3 for the Udacity grading system. Here is how each performed:

The decision tree classifier had a precision of 0.31336 and a recall of 0.59100, both above the 0.3 threshold. The SVM classifier (with features scaled) had a gaudy precision of 0.83333, but a poor recall of 0.06500. The k-nearest neighbors classifier had a precision of 0.32247 and a recall of 0.29200, both near but not both above the 0.3 threshold. As described by Udacity Coach Mitchell, the SVM classifier is not a good fit for the unbalanced classes in the Enron data set, i.e., the lack POIs vs. the abundance of non-POIs. The SVM overfit to the points in the training set, and poorly generalized to the test set resulting in a relatively small number of positive predictions of POIs.

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”]

Tuning the parameters of an algorithm means adjusting the parameters in a certain way to achieve optimal algorithm performance. There are a variety of "certain ways" (e.g. a manual guess-and-check method or automatically with GridSearchCV) and "algorithm performance" can be measured in a variety of ways (e.g. accuracy, precision, or recall). If you don't tune the algorithm well, performance may suffer. The data won't be "learned" well and you won't be able to successfully make predictions on new data.

I used GridSearchCV for parameter tuning. As Katie Malone describes for Civis Analytics, GridSearchCV allows us to construct a grid of all the combinations of parameters, tries each combination, and then reports back the best combination/model.

For the chosen decision tree classifier for example, I tried out multiple different parameter values for each of the following parameters (with the optimal combination bolded). I used Stratified Shuffle Split cross validation to guard against bias introduced by the potential underrepresentation of classes (i.e. POIs).

criterion=['gini', 'entropy'] splitter=['best', 'random'] max_depth=[None, 1, 2, 3, 4] min_samples_split=[1, 2, 3, 4, 25] class_weight=[None, 'balanced'] Note that a few of these chosen parameter values were different than their default values, which proves the importance of tuning.

From the project we go to know how well tuning can affect the performance.

Every algorithm has a set of default values using which the model if fitted and prediction is carried out.Tuning is about setting the right parameters overriding the default parameters in accordance with the requirement of the objective and also ensure that the algorithm is not overly-fitted.

For Decision Trees the parameters were tuned using the param_grid of the GridsearchCv method. The GridSearchCv helps in trying out various hyper parameters combination and picks the bext combination of parameters which is then used to fit the data and for prediction.

Best Parameters:

{'DT__class_weight': 'balanced', 'DT__max_features': 'auto', 'DT__max_depth': 3, 'DT__min_samp les_leaf': 5, 'DT__criterion': 'gini', 'DT__random_state': 42}

For algorithms which do not have parameters:

Certain algorithms like Naive Bayes - Gaussian NB do not have parameter tuning.It can be noted that feeding the algorithm with highly co-related features can add more weights to the same and bring down the performance.Hence some of the highly co-related features were pruned and a subset of the final feature list was passed to this algorithm.Also PCA was used for dimension reduction.Pipelines were used to mention the steps.

Pruning was done outside of pipeline and the steps involved PCA followed by scaling followed by algorithm fit.

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 a way to substantiate your machine learning algorithm's performance, i.e., to test how well your model has been trained. A classic validation mistake is testing your algorithm on the same data that is was trained on. Without separating the training set from the testing set, it is difficult to determine how well your algorithm generalizes to new data.

In my poi_id.py file, my data was separated into training and testing sets. The test size was 30% of the data, while the training set was 70%.

I also used tester.py's Stratified Shuffle Split cross validation as an alternate method to gauge my algorithm's performance. Because the Enron data set is so small, this type of cross validation was useful because it essentially created multiple datasets out of a single one to get more accurate results.

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"]

For this project 2 evaluation metrices were closely monitored to tune and end up with the final choice.
1.Precision
2.Recall
Output from poi_id.py
F1score: 0.664273648649

Total predictions:
11000
True positives:
1433
False positives:
3046
False negatives:
567
True negatives:
5954

The two notable evaluation metrics for this POI identifier are precision and recall. The average precision for my decision tree classifier was 0.31336 and the average recall was 0.59100. What do each of these mean?

Precision is how often our class prediction (POI vs. non-POI) is right when we guess that class Recall is how often we guess the class (POI vs. non-POI) when the class actually occurred In the context of our POI identifier, it is arguably more important to make sure we don't miss any POIs,

so we don't care so much about precision. Imagine we are law enforcement using this algorithm as a screener to determine who to prosecute. When we guess POI, it is not the end of the world if they aren't a POI because we'll do due diligence. We won't put them in jail right away. For our case, we want high recall: when it is a POI, we need to make sure we catch them in our net. The decision tree algorithm performed best in recall (0.59) of the algorithms I tried, hence it being my choice for final analysis.