Updated report: 20 December 2018

Project Documentation

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"]

Answer:

The main goal of this project is to use supervised learning techniques in order to build a classifier that can identify persons of interest (POIs) in the Enron scandal. Enron was an energy, commodities, and services company, which at one time ranked among the largest corporations in the United States. In October 2001, it became the subject of one of the largest cases of accounting fraud. During the subsequent investigation, large amounts of confidential corporate data entered into the public record, which forms the basis of the dataset used in this project. POIs are defined as Enron employees "who were indicted, reached a settlement or plea deal with the government, or testified in exchange for prosecution immunity". Generally speaking, applied machine learning serves two main purposes: prediction and inference. Information from the dataset is used to build a classifier, which can determine whether or not an individual should be considered a POI. This is the primary focus in this project. Moreover, in terms of inference, insights gleaned via machine learning can help uncover relevant drivers for outcomes. In this respect, we can potentially learn something about important factors with respect to involvement in corporate fraud by being able to differentiate between more and less predictive variables. Finally, from the perspective of the Data Analyst Nanodegree, the project allows us to practice the actual end-to-end *process* of machine learning using a real-world dataset.

In contrast to most of the datasets used in the coursework, this dataset has important limitations and imperfections, which is common for real-world data. The two most important challenges are its fairly small size and the relatively large amount of missing data. With respect to the size, we are starting out with 146 total data points. After data cleaning (see below), there are only 144 samples remaining. In addition to the POI labels, the dataset contains information on a total of 21 features. These include features on email (6) as well as financial (14) data. All of the available features have significant amounts of missing values (between 13 and 97 percent). In addition to that, the target labels are unbalanced, meaning there are a lot fewer POIs (18) relative to non-POIs (126). As a first step, some data cleaning had to be performed. Exploratory data analysis (EDA) showed an important outlier named 'TOTAL' with much higher values for all the quantitative features than the rest of the samples', which simply is the aggregation of values across all cases. Since

this is just a spreadsheet quirk, this record was removed. Further manual inspection of the data showed a second record named 'THE TRAVEL AGENCY IN THE PARK', which contained only missing data and appeared to be another quirk (at least it does not seem to represent an actual employee). Consequently it was also removed. Further inspection showed that there are a number of numerical outliers in that certain individuals have much higher values for some of the financial features. However, these are valid and relevant data points, which may hold important information as to whether or not a person is likely a POI. Therefore, all other samples were retained.

The dataset contains the following features:

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)

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"]

Answer:

In the end, I went with a relatively simple model using only 4 features, namely: poi (as the outcome variable), salary, exercised_stock_options, and bonus. Using only these features allowed me to build a more straightforward model with better or at least comparable performance compared to more complex models. This seemed preferable in particular as this is a fairly small dataset with only 144 samples, therefore using fewer predictors seemed particularly important as a guard against overfitting (cf. discussion below). Regarding the feature selection process, initially I

considered all available features. At first, I eliminated the email_address feature as there is no reason why a person's email address itself should have any predictive value. Moreover, I discarded the 'other' feature as it was not clear what exactly this feature represented. Next, I also eliminated all other features that had 50 percent or more missing values. Consequently, the following five features were removed: 'deferral_payments', 'restricted_stock_deferred', 'loan_advances', and director_fees'.

I also engineered and experimented with two new features. Given that the idea seemed intuitively plausible, I re-built the two features suggested in the course, namely the ratios of messages to/from POIs relative to all incoming/outgoing messages ('fraction_from_poi', 'fraction_to_poi'). It seemed indeed that the rate of communication with POIs might be more relevant and predictive than the overall email volumes.

For the actual feature selection and model building process, I relied mostly on manual selection, using my intuition and domain knowledge as well as model performance for guidance. For a much larger dataset, this strategy might have proven unfeasible, but given the relatively small number of features, it seemed perfectly doable to select features by hand and test different combinations guided by intuition.

I started out with a much larger model, using all the remaining variables after the initial filtering step (see above). From there, I gradually tried to drop individual features one by one, retaining only those that truly seemed to carry independent predictive value. While this larger initial model achieved fairly high accuracy (0.86567) and precision (0.48401) using the tester script, its recall remained unsatisfactory (0.11350). Next, I replaced the email features with the two new features as a more parsimonious and relevant representation of the email portion of the data. This led to only slightly lower accuracy (0.86080) and precision (0.44541), but saw a moderate improvement of recall (0.17950). Therefore, for the time being these features were retained in place of the initial five email features. Next, I tried to remove features one-by-one that were highly correlated with other financial features and/or were unlikely to represent features of independent substantive importance. Proceeding in this fashion, I eliminated the following financial features: 'long_term_incentive', 'expenses', 'total_stock_value', and 'total_payments'. After these steps, performance had improved considerably. Now all relevant metrics, i.e., accuracy (0.86077), precision (0.58024), and recall (0.34350), achieved acceptable levels. At this point, further removal of any of the remaining financial features would have led to a drop in the evaluation metrics, therefore I retained 'salary', 'bonus', and 'exercised_stock_options'. Lastly, I tried to ascertain whether the two email features truly had any predictive value. It turned out that their removal barely impacted accuracy (0.86231) and precision (0.58523) while in fact slightly improving recall (0.36050). Therefore, these features were dropped and the final

model only contains bonus, salary, and exercised_stock_options as predictors. While my final chosen algorithm KNN (see below) tends to benefit from using standardized predictors, this was not necessary in this case, as all three predictors are on the same scale. Thus no rescaling had to be performed.

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"]

Answer:

I first used a decision tree classifier as my initial algorithm for two reasons. First, decision trees are able to capture non-linear trends in the data, which I suspected would be the case in this dataset based on the EDA and intuition. Second, among non-parametric methods, decision trees are fairly straightforward and interpretable, therefore it would be beneficial in terms of inference to use a less complex classifier. While I was able to achieve fairly acceptable levels of accuracy, all in all recall and precision remained unsatisfactory using decision trees. This is likely the case because of decision trees' tendency to overfit on the training data. Therefore, I proceeded by testing two more algorithms, namely random forests and k-nearest neighbors (KNN). Random forests tend to achieve higher performance levels than decision trees as they constrain variance. However, just like all ensemble methods they are a more complex and less interpretable type of classifier. KNN on the other hand, is a fairly simple classifier that is quite intuitive to understand and can achieve fairly high performance levels.

It turned out that on this fairly small dataset random forests did not improve performance as much as I had expected. Furthermore, training times were much higher compared to more simple algorithms when using cross-validation techniques. Therefore, I focused on KNN as my final algorithm, which was able to capture the patterns in the data quite well and consequently saw acceptable performance levels after successful tuning and feature selection.

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? (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"]

Answer:

Tuning parameters are quantities that affect how the algorithm learns, i.e., how fast, which parts of the data it tends to "listen" to more closely, etc. Tuning parameters

have to be selected by the researcher, and their values have important consequences for an algorithm's performance. An important concept in this context is the so-called bias-variance tradeoff. Every model has a certain amount of error in the sense of a deviation from the "true" model. It tries to approximate this underlying model as good as possible by "listening" to the data and capturing the main patterns in the form of a function. In this process, one typically encounters two types of error, namely bias and variance. Bias is the amount of systematic deviation from the true patterns. This is due to the fact that every model is an approximation and as such it necessarily misses certain patterns as it is trying to summarize and simplify the underlying context. Variance, on the other hand, is the degree to which the resulting model would change if one were to use different sets of training samples. Every dataset contains both a signal and some noise, i.e., systematic information about underlying relationships as well as random variation. The more literally an algorithm takes the data, or the more closely it tries to "listen", the better it can pick up nuggets of information, yet the more it is also susceptible to simply capture noise. When picking up significant amounts of noise, the algorithm will attune itself too closely to the training data. This will result in high accuracy on the training data itself, but much lower accuracy levels on independent test data. This situation is commonly referred to as overfitting.

A good model will find an acceptable sweet spot between having either too much bias or too much variance by balancing out the inevitable tradeoff. The appropriate selection of tuning parameters is one way of ensuring acceptable performance on independent test data by optimizing how the algorithm learns from the data. Given that my final algorithm is KNN, there is only one core tuning parameter, which is the number k of the nearest neighbors to predict the outcome for a new data point. I used GridSearchCV to find the optimal number of neighbors. I used k=4 folds as 4 is a multiple of the 144 training samples. I tried the following settings: 4, 6, 8, 10. GridSearchCV returned k=4 as the best-performing algorithm, which is what I used for my final model.

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"]

Answer:

Validation requires the researcher to investigate the performance of a model on independent data that was not used during the training process. As explained above, if an algorithm attunes itself too closely to a dataset, it will also capture intricacies and quirks of this particular training sample, which will lead to overfitting and lower performance on new data that the algorithm has not seen before. The easiest way to guard against this is to split the data before model building into a training

and test data. Then, only the training data is used to build the algorithm and the test set is used to assess its performance. A classic mistake would be to use the same data points both for parameter tuning and to evaluate the performance of the algorithm (the real-world performance of the algorithm would likely be overestimated). An alternative is to use cross-validation, where the training data is split into several folds and each fold is once used as the test set while all other folds are used for training. Then, results are averaged to get a sense for how well a model generalizes to new data.

In this project, I made use of the "test_classifier" function provided in tester.py to estimate the average performance of my final model for new samples (i.e. how well it generalizes). This function uses sklearn's StratifiedShuffleSplit to split the data into training and test samples. For the final evaluation of the classifier, 1000 splits were performed (as 1000 is passed to the n_iter argument in tester.py). The StratifiedShuffleSplit randomly splits the data into training and test samples while preserving the percentage of samples for each class (i.e. the proportions of true and false labels from the original data set). This was deemed appropriate in particular given that the data set is fairly small, therefore the information contained in the samples should be maximized while avoiding overfitting the data. In the end, performance on the held out test data (across the 1000 splits) was averaged to get an estimate for how well the classifier generalizes to new data. In my opinion, this was an appropriate way to assess the performance of the final classifier.

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"]

Answer:

The default evaluation metric for a classifier is its accuracy, or the share of cases that it correctly predicts. However, additional evaluation metrics can be more informative in many cases. The two main additional ones used and discussed here are precision and recall. In contrast to accuracy, these two metrics are in reference to the actual outcome one tries to predict, i.e., in relation to POIs. Given that this classifier intends to correctly predict POIs and given that the outcome labels are imbalanced with many more non-POI samples relative to POIs, precision and recall are more relevant metrics when evaluating this classifier.

The two metrics are defined as follows:

Precision:
the number of correctly identified POIs / all cases that were predicted as POI

Updated report: 20 December 2018

Recall:
the number of predicted POIs / all cases that were in fact POIs

In other words, precision controls the rate of false positives (or type I errors) while recall controls the rate of false negatives (or type II errors). In layman's terms, optimal precision requires the classifier to only predict POI when the person actually is a POI; conversely, optimal recall is achieved when all POIs are correctly identified (irrespective of how many additional incorrect predictions there were).

As listed in the table below, the final algorithm (KNN, k=4 with poi as the outcome and salary, bonus, and exercised_stock_options as predictors) had an average precision of 0.59 and an average recall of 0.36. (In addition to that, both the F1 and F2 scores are provided for transparency and completeness. Both are weighted averages of precision and recall.)

Metric	Result
Accuracy	0.86231
Precision	0.58523
Recall	0.36050
F1	0.44616
F2	0.39049