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 goal of this project is to identify fraud in the enron case. Fraud in this case means specifically, that we are asked to identify Persons Of Interest (POI) by using the provided enron dataset(s). Provided are personal and financial information of enron management level employees and their corporate emails in text-files, as well as some already processed information on the emails. Machine Learning can help us in identifying who is a POI and who is not out of the provided samples (persons). We have 145 samples (persons) in the dataset and only 18 are POI. Thus, we have an imbalanced classification task with very few datapoints and our algorithm might have a hard time to identify distinctive patterns that will allow it to properly distinguish a POI from a non-POI. Outliers in the financial data were processed using a function and the effect of their removal on classification (precision and recall metrics used) was evaluated by using cross validation. We removed two outliers as specified in the code and this improved cv scores for us.

1. 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”, “intelligently select features”, “properly scale features”]

Answer: we first min max scaled the data to allow for comparability of features and then created interaction features which we rigorously selected for their cross validation score. Scaling was important so as not to give too much weight to high-number variables like salary (thousands) as opposed to number of emails received from a POI (less than a thousand). Their unscaled combination would result in unbalanced interaction variables with the former variables influence dominating. The rationale behind creating interaction variables (and others) is that we can find patterns in the data that were previously hidden to our algorithm and maybe improve our cross validated classification results. We did not use any features created from email-text. We ended up using the 299 features specified in the code file and applied PCA on it. This was the best f1-score result so we went with that rather than SVC with 43 features and a little worse f1-score.

For the SVC we used an iterative process to identify which and how many features to use by using SKB and PCA (around 300 created features which we did not remove initially) to maximize CV score and minimize overfitting to too many poorly understood features. We tried to use the least amount of features for the best CV result. Scaling with Standardscaler was necessary for some algorithms we used, like SVMs. We also applied an oversampling method called SMOTE which yielded amazing results compared to the dataset with feature engineering but without oversampling (f1 and f2 scores below 0.55 to above 0.95). The problem with SMOTE is that depending on the initialization we get a difference of almost 2 percent in f1 score. For the final result we used a random\_state that gave us a 0.97 f1-score.

1. 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: we tried LogisticRegression, SVMs, RandomForestClassifier, GaussianNB and XGBClassifier. Finally we decided on LinearSVC because of best performance in CV. Model performances for best found parameters in GridSearchCV were:

SVC -f1-test-score : 0.97. Logistic Regression has also a good performance. RandomForests, XGB, SVC, KNN all performed well in initial setup with our dataset with interaction features and the oversampling. GaussianNB had a great performance as well and does not need to be (can’t be) tuned. See table in code file.

1. 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 items: “discuss parameter tuning”, “tune the algorithm”]

Answer: especially algorithms that can fit the training data extremely well can overfit your training data. This goes for DecisionTree based algorithms like RandomForest and XGB. Overfitting entails that we predict poorly on unseen (test) data and we therefore have a high variance, which is undesirable. Overfitting is not as much a risk for LogisticRegression since it is a parametric method and parameters are evaluated by the algorithm during training where less impactful variables are given less weight. But poorly structured data can influence the LR results, multicollinearity can be a problem. We tuned our final candidate algorithms (SVC and LinearSVC) with gridsearchCV to minimize overfitting and underfitting. Most important parameters are the number of selected features/dimensionality reduction method, C, gamma, class\_weight, tol.

Parameter tuning is an important step and can improve algorithm performance on CV results and test data. On the other hand it can cost a huge amount of time to find the perfect or best combination for a given dataset (features and sample size). We invested a lot of time into fine tuning our algorithms and opposed to that GaussianNB performs great (best on f1 score) without much time invested other than picking out number of features or components with CV.

1. 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 items: “discuss validation”, “validation strategy”]

Answer: validation is necessary to gauge our models performance on previously unseen data not used in testing or model creation. A mistake we can make is to use all the data we have to train our algorithm and overfit the training data, leading to higher error when predicting on unseen data (high variance). The simple version is to split your training set into training and validation sets. A more sophisticated approach that costs more time but might yield better results is to do “everything” with a cross validation approach in a pipelined environment. Everything means: feature and model selection, imputation strategies, outlier removal and hyperparameter tuning. The idea behind cross validation is that we use our whole training set to decide on features and algorithms to use for our final model that will predict on the unseen test data. We can avoid overfitting to our train set by splitting the training set n times into (n-1)-folds of combined training set and 1 fold of a validation data set. For every nth fold we use just this one fold as a validation set which we do not use to train the model on and the other folds as the training set for our model. We repeat this process n times and average our results on the individual validation folds to get the final estimate of our models performance on unseen data. The more similar our train and unseen test set are, the better the predictions of our final model. Depending on the size and type of data used for training, this can yield very accurate results. We used 10-fold stratified CV to make informed decisions about which features to use and which algorithm / hyperparameters to pick for our final model. Stratified means that we made the folds by preserving the percentage of samples for each class in every fold.

1. 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: for an imbalanced classification task accuracy is not an ideal metric. We might have only 2 percent of one class in our dataset, always predicting one class will yield very good accuracy, because we are 98% correct/accurate all the time. But this is not what we want, we want to detect these occurrences correctly and therefore we need better metrics to gauge our performance on that goal. We use precision (number of correctly classified as belonging to the minority class samples vs. total number of “marked as belonging to minority class” samples by our model) and recall (how many samples truly belonging to the minority class were identified by our model vs total number of minority class samples in our test set). High precision means that we have a low false positive rate, we are very conservative or very precise in classifying sb. as a POI. High recall means that we try to find every POI in the dataset and risk a high false negative rate because we misclassify some as POI when they are actually not. These two metrics can be combined to the f1-score which is a weighted average of these individual scores. Higher precision usually means that we have lower recall and vice versa. We got a final score (tester.py) of 0.999 precision and 0.945 recall, which means that we can correctly classify employees as POI 99.9% of the time and if there are POIs in the dataset that we can identify 94.5% of them. This combines to a f1-score of: 0.971. Therefore, our algorithm is good, but a little unbalanced. It is more precise in correctly identifying POIs as opposed to finding all the POIs in the data. It rather misses some than classifies edge cases as POI which are actually not.