SUMMARY

JESUA EPEQUIN

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?

The goal of this project if to set up an algorithm allowing us to discern whether an employee of Enron was or not part of the fraud scandal that took place at the turn of the 21st century.

After the collapse of the Enron corporation in 2002, and during the resulting legal investigation by the Federal Energy Regulatory Commission, a database was generated from the Enron's enterprise database systems and email servers. At the conclusion of the investigation the emails and information collected were deemed to be in the public domain, to be used for historical research and academic purposes. This is how the *Enron corpus* (one of the few publicly available mass collections of real emails) was born.

For this project, a summary of the email and financial data obtained from the corpus is given as a dictionary data_dict. It is contained in the pickle file final_project_dataset.pkl. The keys of this dictionary are strings corresponding to the names of 146 employees. The values for these keys are dictionaries containing relevent information concerning the given person: salary, bonus, total stock value, number of sent emails, number of received emails, etc. In addition to the email and financial information, we can find a key-value pair telling us whether this person is a POI (person of interest) of not. This information makes it possible for us to use supervised learning to achieve the goal of detecting POIs.

During the data exploration we found three outliers in our dataset, they correspond to the following keys:
- 'TOTAL': it contained aggregate of features for all rows.
- 'THE TRAVEL AGENCY IN THE PARK': clearly not an employee.
- 'LOCKHART EUGENE E': it had 'NaN' for all features.

After removing these outliers, the dataset consisted of 143 rows. Of these, 125 were labelled as Non-POIs, the remaining 18 as POIs.

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.

We implemented two new features:

a) 'percentage_shared_with_poi': this feature encodes the percentage of incoming emails that were also received by a POI.

b) 'percentage_sent_to_poi': this feature stands for the percentage of sent messages that were addressed to a POI.

The rationale behind these two features was that POIs might have communicated more among each other that with the rest of the employees on the dataset. A simple visualization showed that this is indeed the case (see Figure **??**):

a) If 'percentage_shared_with_poi' has a value inferior to 40% then employee is likely not a POI.

b) If 'percentage_sent_to_poi' is below 20% then the person is likely not a POI.

Since we decided to test different several algorithms (not including decision trees), univariate feature selection was deployed. We used the SelectKBest function from sklearn module to decide which were the features with the hightest score (for the default f_classif function). It turns out that optimal results are obtained with $k = 5$. The selected features and their scores are shown in the following table:

We note that one of our engineered features ('percentage_sent_to_poi') was used, and that it is considerably stronger than the two features used to create it: 'from_messages' which had a score of 0.17, and 'from_this_person_to_poi' with a score of 2.38. This is intuitive because:

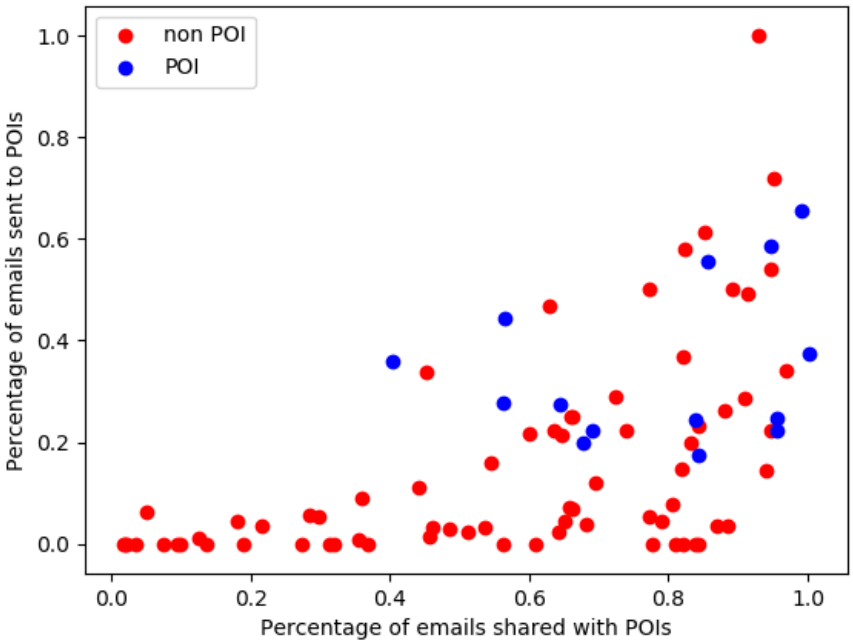

FIGURE 1. Shared with POIs % vs Sent to POIs %

Feature	**Score**
'exercised_stock_options'	24.82
'total_stock_value'	24.18
'bonus'	20.79
'salary'	18.29
'percentage_sent_to_poi'	16.41

a) There is no reason why the number of sent messages should be correlated to being part of the fraud.

b) The number of messages sent to POIs does not necessarily mean that a person had a role in the fraud. Indeed, it is reasonable to assume a direct relation between the total number of sent messages, and the number of those addressed to POIs (people sending a higher number of emails have a higher chance of also sending more messages to POIs).

However, the ratio of these two numbers provides better information, as explained above. Selecting these five features, we obtained the highest values for the metrics used to evaluate the performance of our classifier (see below).

We did not perform any scaling because the algorithm that provided the best results among those we tested (random forest) is not sensitive to it. We did not use dimensionality reduction either, the number of features was already small enough.

3. What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms?

As part of the project, we let the user choose one of three algorithms: Support Vector Machines, Gaussian Naive Bayes, or Random Forests. We show the accuracy of these in the following table:

Model	Accuracy
Support Vector Classifier	84%
Gaussian Naive Bayes	79%
Random Forest	90%

Looking at the accuracy we could say that Random Forests make fairly good predictions. However, this metric is particularly misleading for imbalanced datasets. As we saw above, of the 143 data points in our dataset only 18 (that is 13%) are labelled as POIs: an algorithm assigning every single case to the non-POIs class (not a very intelligent approach) would have a 87% accuracy. Below we will discuss better metrics for this dataset.

Out of these three classifiers, the most performant (for all metrics considered) is the Random Forest. In general, Decision Trees (and hence Random Forests) perform well on imbalanced data, especially if the minority class is in one area of the feature space.

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).

Parameter tuning is an important step after selecting our algorithm. Failing to do this can produce problems like under or overfitting. Consider for instance the *min_sample_split* parameter of a decision tree.

Low values can lead to overfitting (because of high complexity of the decision boundary), higher values prevent a model from learning relations which might be highly specific. However, too high values can lead to under-fitting.

In order to tune our parameters, we could go read research papers on our classifier and try to theorize the best among the many available parameters. However, a more efficient use of our time is just to try out a wide range of values and see what works. We used the GridSearchCV function from the sklearn module to test different combinations of parameters (instead of doing it one by one). We trained the following parameters:

a) For the Support Vector Classifier: *kernel* (values: 'rbf', 'sigmoid'), *C* (values: 1/10, 1, 10), *gamma* (values: 'scale', 'auto', 1/10, 1, 10).

b) For the Random Forest Classifier: *n_estimators* (values: 10, 100, 1000), *min_samples_split* (values: 2, 5, 10), *max_depth* (values: 10, 50, 100), *criterion* (values: 'gini', 'entropy')

5. What is validation, and what's a classic mistake you can make if you do it wrong? How did you validate your analysis?

Validation is the process of evaluating an already trained model on test data set. This is done to assess our model performance when confronted to unseen data. It is a crucial step in any machine learning project.

One common mistake is to use the whole data set to train our algorithm. We could be tempted to do so, because of the high accuracy of prediction the ensuing model would usually have. However, it is unwise to use all the data points as training set because it results in overfitting. A model obtained in this way would not generalize well to new data.

Therefore, it is important to split our dataset in two parts: train and test sets. We use the train set to fit our algorithm (so it creates a decision boundary), and then we test its accuracy (and other metrics) on the test set. Normally the value for these metrics drop if we validate on the test set instead than on the train set, but this is normal (we should always be suspicious of metric values that are too high).

In order to split our dataset we use the train_test_split function from the sklearn module. This function has two important hyperparameters that we tune for better results: *shuffle* and *stratify*.

a) Shuffling is important because we want to make sure that there is enough variety in our training set for our model to create an appropiate

decision boundary. Indeed, imagine a extreme case where the first 50% of our data set is labelled as 'A', while the remaining 50% belongs to the class 'B'. If we were to split our data set in two equal parts and use the first one (labelled 'A' data samples) to train our algorithm, then we would obtain poor results after testing it on the second one (labelled 'B' data samples). This is reasonable since our model never encountered data points of the second class during the training phase.

b) Stratifying is done so the train and test sets have the same proportion of POIs to non POIs. This is complementary to the shuffling process, and insures that its purpose (explained above) is achieved.

We can insure that shuffling is performed by passing the value True to the hyperparameter *shuffle* (this is actually the default), and passing an the whole array of labels to the parameter *stratify*. We also fix the train set size to 67% and the test set to 33% of the whole dataset.

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 algorithms performance.**

As mentioned above, accuracy is not the best metric for an imbalanced dataset. During the course we were introduced to two metrics that convey different information, and that prove more meaningful for this particular project: precision and recall. They are defined as follows:

$$\text{precision} = \frac{\text{True positive}}{\text{True positive} + \text{False positive}}$$

$$\text{recall} = \frac{\text{True positive}}{\text{True positive} + \text{False negative}}$$

We see that both these metrics have values in the interval $[0, 1]$. For an explanation in plain terms we quote the sklearn documentation page:

- "The precision is intuitively the ability of the classifier not to label as positive a sample that is negative".

- "The recall is intuitively the ability of the classifier to find all the positive samples".

So, for this project high precision values mean that we can be confident that most of the predicted POIs are indeed POIs. High values for recall mean that our algorithm is good at finding POIs.

A third metric we tried is the F_1-score. It is defined as the harmonic mean of precision and recall:

$$F_1 = 2\frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}.$$

The F_1-score is a number contained in the interval $[0, 1]$. It reaches its optimum 1 only if precision and recall are both at 100%. And if one of them equals 0, then this score has its worst value 0. The F_1 score is one of the most-recommended quality measure for binary classification problems. In this project, we choose the hyperparameters for our classifier so that F_1-score is as high as possible. We do this by setting the *scoring* parameter of GridSearchCV to the value 'f1'.

The F_1, precision, and recall score values for the algorithms we tested are presented in the table below.

Model	$\mathbf{F_1}$	**Precision**	**Recall**
Support Vector Classifier	0.22	0.2	0.25
Gaussian Naive Bayes	0.18	0.2	0.17
Random Forest	0.5	0.4	0.67

Again, the Random Forest Classifier is the most perfomant. We obtain a precision of 40% and a recall of close to 70%. Having a high value for recall is more important than having one for precision in this project. Indeed, we would like our model to predict as many POIs as possible, because this would increase our list of *potential* culprits. It is ok if some of these were not indeed part of the fraud because each of these gets to have a trial (a second and final filter to decide if they are guilty or not).