PROJECT REPORT

BY

SAAD KHAN

Table of Contents

1 QUALITY OF CODE

1.1 CODE FUNCTIONALITY

Code written for this project reflects the description in the documentation. A combination of the code implemented in the various mini projects at the end of lessons was used for the project.

1.2 CODE USABILITY

The dataset, feature list and algorithm were exported using the poi_id.py program. These exports can be verified using the tester.py program. Before the creation of final poi_id.py program the code was written in Ipython notebook for testing purposes (included in the github repo).

2 INTRODUCTION

Enron Corporation was an American energy, commodities and services company based in Houston, Texas. Before its bankruptcy on December 2, 2001, Enron employed approximately 20,000 staff and was one of the world's major electricity, natural gas, communications, and pulp and paper companies, with claimed revenues of nearly $111 billion during 2000. Fortune named Enron "America's Most Innovative Company" for six consecutive years.

The Enron scandal, revealed in October 2001, eventually led to the bankruptcy of the Enron Corporation, an American energy company based in Houston, Texas, and the de facto dissolution of Arthur Andersen, which was one of the five largest audit and accountancy partnerships in the world. In addition to being the largest bankruptcy reorganization in American history at that time, Enron was cited as the biggest audit failure.

In the resulting Federal investigation, there was a significant amount of typically confidential information entered into public record, including tens of thousands of emails and detailed financial data for top executives.

3 UNDERSTANDING THE DATASET AND QUESTION

NOTE: This section also addresses question 1 from the free-response questionnaire.

Question 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?***

3.1 PROJECT GOAL

The goal of the project is to build a person of Interest (POI) identifier, using various machine learning techniques, based on the email messages and financial data made public as a result of the Enron scandal. POIs in this case are people who worked at Enron and were probable suspects for the fraudulent activities performed there which eventually led to its bankruptcy.

Machine learning will help predict if a certain person picked at random who worked at Enron was involved in the corporate fraud or not, i.e. if that person is a person of interest (POI) or not. Concepts of machine learning are to be applied to the available dataset and a POI classifier is to be created which would help predict person of interest.

3.2 DATA EXPLORATION

Some of the important characteristics of the dataset are as follows:

3.2.1 Dimensions of the Dataset (Total no. of points)

```
print 'Total number of observations/data points in the data set:', len(data_dict)
```

```
Total number of observations/data points in the data set: 146
```

```
print 'Total number of features available per observation:', sum(len(v) for v in data_dict.itervalues())/len(data_dict)
```

```
Total number of features available per observation: 21
```

3.2.2 Allocation of observations (POIs vs. Non-POIs)

```
count = 0
for k, v in data_dict.iteritems():
    if(data_dict[k]["poi"]==1):
        count +=1
    else:
        continue
print 'Number of obervations that have POIs:', count
print
print 'Number of obervations that do not have POIs:', len(data_dict) - count
```

```
Number of obervations that have POIs: 18

Number of obervations that do not have POIs: 128
```

3.2.3 Salary and Total payments of some high ups @ Enron

```
print'LAY KENNETH L, Salary:',data_dict['LAY KENNETH L']['salary'],',Total payments:',data_dict['LAY KENNETH L']['total_payments']
print
print'SKILLING JEFFREY K, Salary:',data_dict['SKILLING JEFFREY K']['salary'],', Total payments:',data_dict['LAY KENNETH L']['total_payments
print
print'FASTOW ANDREW S: Salary,',data_dict['FASTOW ANDREW S']['salary'],', Total payments:',data_dict['LAY KENNETH L']['total_payments']
```

```
LAY KENNETH L, Salary: 1072321 ,Total payments: 103559793

SKILLING JEFFREY K, Salary: 1111258 , Total payments: 103559793

FASTOW ANDREW S: Salary, 440698 , Total payments: 103559793
```

3.2.4 Salaried people @ Enron

```
salaried_people = 0
for k,v in data_dict.iteritems():
    if (data_dict[k]["salary"]!= 'NaN'):
        salaried_people += 1
    else:
        continue
print 'Number of people salaried @ Enron:', salaried_people
```

```
Number of people salaried @ Enron: 95
```

3.2.5 Missing values for the Features

```
missing_values = {}

for v in data_dict:


    for l,m in data_dict[v].iteritems():
        if m == 'NaN' and l in missing_values:

            missing_values[l] += 1

        elif m == 'NaN' and l not in missing_values:
            missing_values[l] = 1
pp.pprint(missing_values)
```

```
{'bonus': 64,
 'deferral_payments': 107,
 'deferred_income': 97,
 'director_fees': 129,
 'email_address': 35,
 'exercised_stock_options': 44,
 'expenses': 51,
 'from_messages': 60,
 'from_poi_to_this_person': 60,
 'from_this_person_to_poi': 60,
 'loan_advances': 142,
 'long_term_incentive': 80,
 'other': 53,
 'restricted_stock': 36,
 'restricted_stock_deferred': 128,
 'salary': 51,
 'shared_receipt_with_poi': 60,
 'to_messages': 60,
 'total_payments': 21,
 'total_stock_value': 20}
```

3.3 Outlier Investigation (Detection and Removal / Handling)

Outliers in the data set can result from some malfunction or data entry errors. I analyzed the financial data document 'enron61702insiderpay.pdf' with naked eye without any code and following are the observations that I think are outliers and should be cleaned away as they do not contain any essential information. I handled the outliers using the piece of code I applied in lesson 5 exercise in the format: 'dictionary.pop (key, 0)'.

Following are the outliers removed from the dataset:

- Wendy Grahm only had the feature of director's fees so I thought to remove this observation.
- Another obvious choice for outlier removal was Eugene Lockhart, did not have value associated with any feature.
- Bruce Wrobel, like Wendy Grahm only had one feature, exercised stock options, so I removed it.
- THE TRAVEL AGENCY IN THE PARK did not seem to be a POI.
- TOTAL is not a particular POI as it was the total of all financial features.

Apart from the outlier removal, there were a few records that had discrepancies as per the link from the Udacity discussion forums

http://discussions.udacity.com/t/two-records-financial-values-out-of-sync/8687

I updated the following records that were incorrect/inconsistent.

- data_dict['BELFER ROBERT']
- data_dict['BHATNAGAR SANJAY']

4 OPTIMIZED FEATURE SELECTION/ENGINEERING

NOTE: This section deals with the importance of features in the dataset and also answers question 2 from the free-response questionnaire.

Question 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 doesn't 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.) If you used an algorithm like a decision tree, please also give the feature_importance's of the features that you use.***

I started off by testing combinations of features for various algorithms. The performance of each algorithm for each feature combination was recorded and used to identify the best performing algorithm (this is covered in section 7). Once the best performing algorithm was identified it was then tested again with different feature combinations to find the best possible feature set for final analysis (all this is covered in later sections).

This section covers the features that I tested for the initial identification of the best performing algorithm. Tested features were a combination of the following:

- Original features in the dataset
- New features created from the original email features
- Emails word data used as features

Feature selection and scaling was also performed while creating the testing scenarios for different algorithms. Details are covered in the section 7.

Aspects covered in this section are as follows:

Creation of new features:

- 2 new feature created from the original features in the dataset
- Several text features created from the email archives

Intelligent selection of features for algorithm testing:

Selection of features for algorithm testing was a combination of the following:

- Hand picking the features from the original dataset through intuition
- Using newly created features in combination with the original feature set
- Performing selection techniques from the sklearn library (SelectKBest, SelectPercentile, etc)

Proper scaling of features (if required):

Details of feature scaling are covered in section 4.4

4.1 Initial Feature Selection for Algorithm Performance Testing

The initial feature selection for algorithm testing was done by hand picking/intuition from the original dataset. List of all 21 original features in the dataset are as follows:

```
pp.pprint(data_dict['LAY KENNETH L'].keys())
```

```
['salary',
 'to_messages',
 'deferral_payments',
 'total_payments',
 'exercised_stock_options',
 'bonus',
 'restricted_stock',
 'shared_receipt_with_poi',
 'restricted_stock_deferred',
 'total_stock_value',
 'expenses',
 'loan_advances',
 'from_messages',
 'other',
 'from_this_person_to_poi',
 'poi',
 'director_fees',
 'deferred_income',
 'long_term_incentive',
 'email_address',
 'from_poi_to_this_person']
```

Multiple features were tested in order to test the performance of different algorithms. The objective was to record relative performance of each algorithm with changing set of features and then choosing the best performing algorithm for the final analysis. There were 3 feature combinations tested against all the algorithms. Testing of all these combinations is covered in section 7.The combinations are as follows:

4.1.1 Original Set of Features

```
features_list =  ['poi', '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',
                  'to_messages', 'from_poi_to_this_person', 'from_messages', 'from_this_person_to_poi',
                  'shared_receipt_with_poi']
```

NOTE: Only feature excluded from the original data set was ‘email_address’

4.1.2 Selected Set of Features

```
features_list =  ['poi', 'salary', 'total_payments', 'loan_advances', 'bonus',
                  'restricted_stock_deferred', 'total_stock_value', 'expenses',
                  'exercised_stock_options', 'long_term_incentive', 'restricted_stock',
                  'to_messages', 'from_poi_to_this_person', 'from_messages',
                  'from_this_person_to_poi', 'shared_receipt_with_poi']
```

NOTE: The selected set of features chosen for algorithm testing was purely through intuition. Here I chose to eliminate some of the features that had no entries for most of the observations according to the output of section *3.2.5 ‘Missing values for the Features’,* following features were left out.

```
'deferral_payments': 107,
'deferred_income': 97,
'director_fees': 129,
```

4.1.3 More Selected Set of Features

```
features_list =  ['poi', 'salary', 'loan_advances', 'bonus', 'total_stock_value', 'exercised_stock_options',
                  'to_messages', 'from_messages', 'shared_receipt_with_poi']
```

NOTE1: Again, the more selected set was chosen to tabulate the performance of each algorithm. This feature list was created also based on intuition.

NOTE2: Feature list selection for the final analysis is covered in the later sections.

4.2 Investigation/Relationship of some of the features that were selected by hand

Few of the features selected from the dataset were plotted to have a general idea about the distribution. Data points marked with red crosses are POIs.

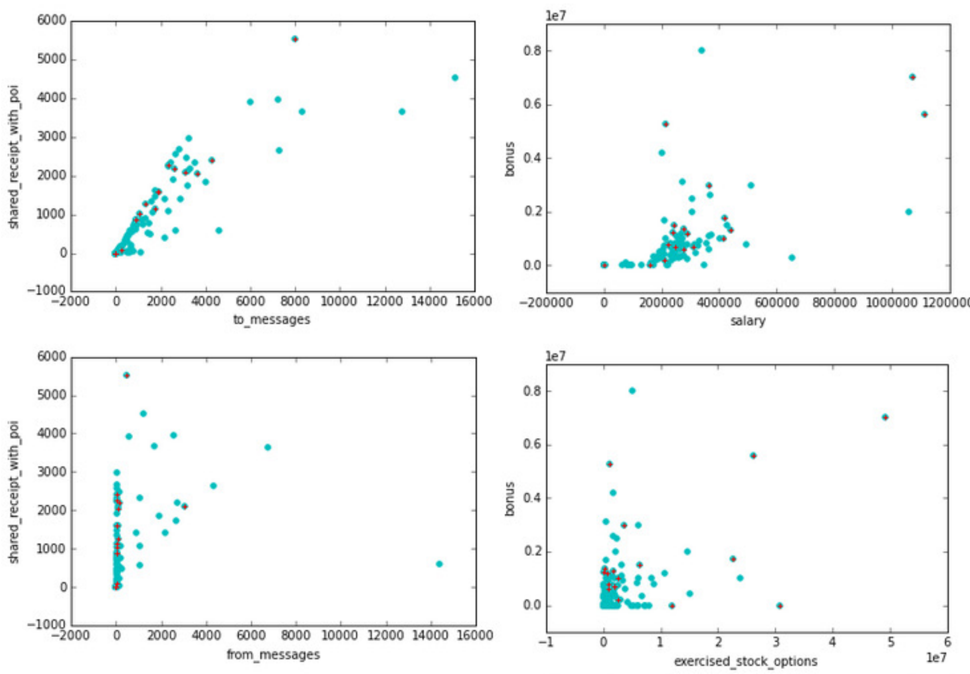


Figure 4.1

4.3 New Feature Creation

4.3.1 Fraction to/from POI emails (2 features) - Combination of original features in the dataset

Using the exercise quiz in lesson 11, a piece of code, 'computeFraction' was implemented to include 2 new features in the data set namely 'fraction_from_poi' and 'fraction_to_poi'. These features are a combination of 4 original features in the dataset.

After creation, features were then stored in the modified feature list, 'my_feature_list' and values for all observations stored in modified data set, 'my_dataset'.

```
def computeFraction(poi_messages, all_messages):
    """ given a number messages to/from POI (numerator)
        and number of all messages to/from a person (denominator),
        return the fraction of messages to/from that person
        that are from/to a POI
   """


    ### you fill in this code, so that it returns either
    ###     the fraction of all messages to this person that come from POIs
    ###     or
    ###     the fraction of all messages from this person that are sent to POIs
    ### the same code can be used to compute either quantity

    ### beware of "NaN" when there is no known email address (and so
    ### no filled email features), and integer division!
    ### in case of poi_messages or all_messages having "NaN" value, return 0.
    if (poi_messages == 'NaN') or (all_messages == 'NaN'):
        fraction = 0.
    else:
        fraction = float(poi_messages)/float(all_messages)

    return fraction


for name in data_dict:

    # print name
    data_point = data_dict[name]
    from_poi_to_this_person = data_point["from_poi_to_this_person"]
    to_messages = data_point["to_messages"]
    fraction_from_poi = computeFraction(from_poi_to_this_person, to_messages)
    # print fraction_from_poi

    from_this_person_to_poi = data_point["from_this_person_to_poi"]
    from_messages = data_point["from_messages"]
    fraction_to_poi = computeFraction(from_this_person_to_poi, from_messages)
    # print fraction_to_poi

    ### Adding values of new features to the modified dataset, 'my_dataset'

    my_dataset[name]['fraction_from_poi'] = fraction_from_poi
    my_dataset[name]['fraction_to_poi'] = fraction_to_poi
```

Following code was written to plot a scatter diagram to examine the distribution of the 2 new features created.

```
poi  = "poi"

poi_fraction_features = [poi, 'fraction_from_poi', 'fraction_to_poi']

data = featureFormat(my_dataset, poi_fraction_features)
poi, testing_fraction_features = targetFeatureSplit(data)

for ii, pp in enumerate(testing_fraction_features):
    if testing_fraction_features[ii][0] != 0.0 or testing_fraction_features[ii][1] != 0.0:
        plt.scatter(testing_fraction_features[ii][0], testing_fraction_features[ii][1], color = 'c')
        plt.xlabel('Fraction of emails this person gets from POI')
        plt.ylabel('Fraction of emails this person sends to POI')
    else:
        continue
for ii, pp in enumerate(testing_fraction_features):
    if poi[ii] and (testing_fraction_features[ii][0] != 0.0 or testing_fraction_features[ii][1] != 0.0):
        plt.scatter(testing_fraction_features[ii][0], testing_fraction_features[ii][1], color = 'r', marker="+")

plt.show()
```

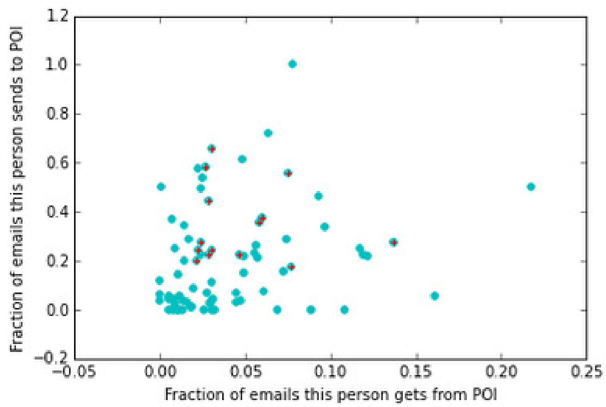


Figure 4.2

The scatter plot represents the fraction of email exchange between Non-POI/POI. Red crosses show the POIs.

Justification: These features can be useful in further classifying POIs. This can be observed in figure 4.2 that there is a definite trend that POIs mostly send/receive emails to/from other POIs. People that are non-POI tend to send fewer emails as can be seen mostly clustered with fractions closer to 0.

The performance of these fraction features is tested in section 7 of this report.

4.3.2 Word features from text in the email archives

Function 'parseOutText' was used from the lesson 10 mini-project along with a modified version of the code used to extract email text in lesson 10 mini-project. First it was checked if the observation had valid email address and then for a valid email address, email text was extracted using the 'parseOutText' function. The snowball stemmer for English is already implemented in the 'parseOutText' function. The text features data was extracted and stored in a dictionary, 'word_data'.

Function 'parseOutText'

```
def parseOutText(f):
    """ given an opened email file f, parse out all text below the
        metadata block at the top
        (in Part 2, you will also add stemming capabilities)
        and return a string that contains all the words
        in the email (space-separated)

        example use case:
        f = open("email_file_name.txt", "r")
        text = parseOutText(f)

        """


    f.seek(0)  ### go back to beginning of file (annoying)
    all_text = f.read()

    ### split off metadata
    content = all_text.split("X-FileName:")
    words = ""
    if len(content) > 1:
        ### remove punctuation
        text_string = content[1].translate(string.maketrans("", ""), string.punctuation)

        ### project part 2: comment out the line below
        #words = text_string

        ### split the text string into individual words, stem each word,
        ### and append the stemmed word to words (make sure there's a single
        ### space between each stemmed word)

        #print text_string

        email_split = text_string.split()

        #print email_split

        from nltk.stem.snowball import SnowballStemmer

        stemmer = SnowballStemmer("english")

        email_stemmed = []
        for n in range(len(email_split)):
            email_stemmed.append(stemmer.stem(email_split[n]))

    words = ' '.join(email_stemmed)

    return words
```

Below is the piece of code to identify valid email address

```
valid_email_id_list = []

email_address_path_folder = os.listdir("../final_project/emails_by_address/")

test_dataset = my_dataset
#observation_removal = []

count1 = 0
count2 = 0

for name in my_dataset:
    #k = name
    email_name = test_dataset[name]
    email_address = email_name["email_address"]
    email_address_path = "from_" + email_address + ".txt"
    #print email_address, email_address_path
    if email_address == 'NaN':
        #observation_removal.append(name)
        #print 'TO BE REMOVED_1:' ,email_address_path
        count1 += 1

    elif email_address_path not in email_address_path_folder:
        #observation_removal.append(name)
        #print 'TO BE REMOVED_2:' ,email_address_path
        count2 += 1
    else:
        valid_email_id_list.append(email_address)
        #print email_address_path


print 'size of the dataset:', len(my_dataset)
print
print 'No of observation that have no email data for word features:', count1 + count2


'''
for n in observation_removal:
    test_dataset.pop(n, None)
'''

my_dataset = test_dataset
```

Final piece of code to parse the text of emails

```
from_data = []
word_data = []

import os
import pickle
import re
import sys
import string


temp_counter = 0

for n in range(len(valid_email_id_list)):

    email_address_folder = "../final_project/emails_by_address/from_" + valid_email_id_list[n] + ".txt"

    #text_file = os.listdir("../final_project/emails_by_address/")

    email_file_path = open(email_address_folder, "r")

    for path in email_file_path:
        ### only look at first 200 emails when developing
        ### once everything is working, remove this line to run over full dataset
        temp_counter += 1
        if temp_counter < 200:


            path = os.path.join('..', path[:-1])
            #print path
            email = open(path, "r")

            stemmed_email = parseOutText(email)

            word_data.append(stemmed_email)
            '''
            if (name == "chris"):
                from_data.append(1)
            else:
                from_data.append(0)
            '''
            from_data.append(n)

            email.close()
```

Justification: In my opinion, features extracted from text emails can be very useful to predict the person of interest. Using words in emails as features would add an additional dimension to the model in creating the POI identifier.

The performance of these text features is tested in the later sections of this report as part of the algorithm selection process.

4.4 Scaling of Features

There were 5 algorithms tested before selecting the algorithm for the final analysis. This section briefly explains which algorithms requires feature scaling and which do not and was it deployed in the final analysis or not.

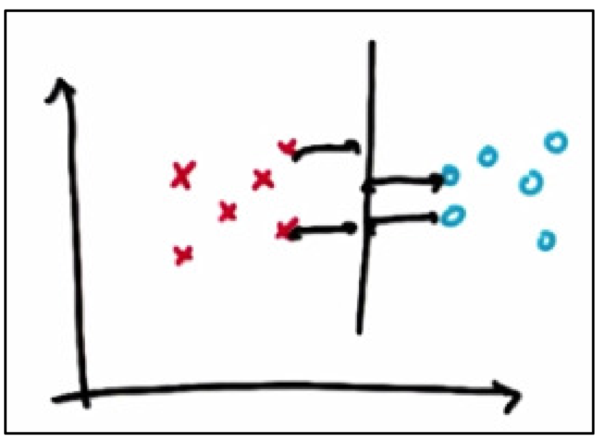
4.4.1 Gaussian Naïve Bayes (feature scaling = NO)

This algorithm does not require feature in my opinion. It is evident from the name 'Gaussian' that features are normalized as part of computation. So, I think applying feature scaling would not be of any additional benefit. This is also evident by the Bayes rule conditional probability calculations performed in lesson 1.

Reference: *Intro to Machine Learning – Lesson 1*

4.4.2 Support Vector Machine (feature scaling = YES)

Support Vector Machine (SVM) does get effected by feature scaling and it is beneficial to use it when deploying SVM. For example, we have our features plotted on scatter plot in 2-dimensions and a separating line is drawn to maximize the distance for applying SVM.

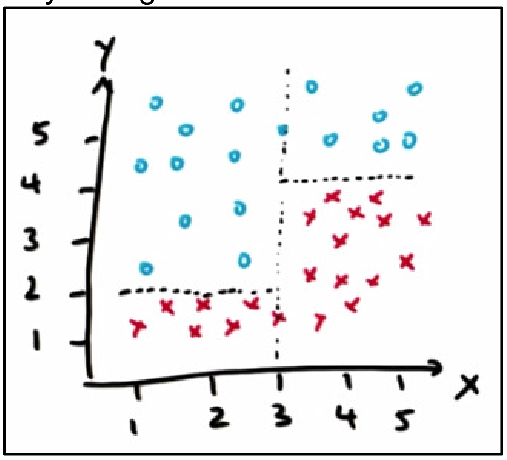


As SVM inputs the features of different classes and outputs a line which maximizes the distance/margin, as shown in the figure above, if one of the feature is twice the dimension of the other one then it would count for as twice as much in magnitude and would shift the output line that maximizes the distance, hence effecting the performance of the algorithm. So in this case, deploying feature scaling can be very useful and would help make the features comparable. The feature scaling applied for SVM for algorithm analysis was MinMaxScaler from the sklearn library in python.

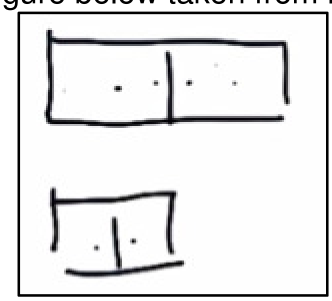
Reference: *Intro to Machine Learning – Lesson 2 and Lesson 9*

4.4.3 Decision Tree Classifier (feature scaling = NO)

Decision trees unlike support vector machines, do not output lines that maximize distance from plotted points rather It outputs certain number of horizontal or vertical lines called decision boundaries depending on the complexity of the plotted features. These decision boundaries do not necessarily have to be at maximum distance from any given point as depicted by the figure below.



So change of scale for any one feature might expand, contract or shift the decision boundary individually but would not effect a feature in the other dimension as the decision boundary separation would essentially scale itself along with the scaled feature. This is depicted in the figure below taken from lesson 9.



Here the decision boundary also scales itself relative to the scaled feature and does not impact the unscaled feature, hence, no feature scaling is necessary.

Reference: *Intro to Machine Learning – Lesson 3 and Lesson 9*

4.4.4 Random Forest Classifier (feature scaling = NO)

Random Forest classifier is part of the ensemble methods and is covered in the sklearn library ensemble methods class. In this algorithm, we do not require any coefficients, which measure the relationship between each predictor feature and the response, for example as is the case with linear regression. In fact, as the name suggests, random forest does classification on the basis of decision trees created with random subsets of observations. So feature scaling is not really required based on the same principal applied to decision tree classifier. Other reason of not deploying scaling is that it can smooth out the nonlinear nature of the model. If you have complex nonlinear relationships in a large dimensional space and you have transformed your data, then when back transforming these nonlinearities will not be depicted correctly.

Reference: *Intro to Machine Learning – Lesson 4*

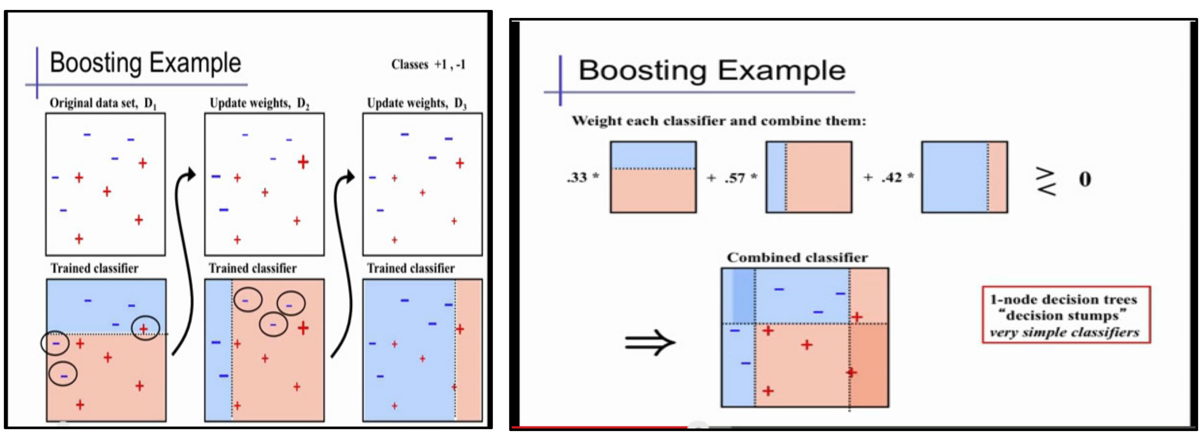
http://stackoverflow.com/questions/8961586/do-i-need-to-normalize-or-scale-data-for-randomforest-r-package

https://www.youtube.com/watch?v=loNcrMjYh64

4.4.5 Adaboost Classifier (feature scaling = NO)

Just like random forest classifier, Adaboost is part of the ensemble methods class of the python sklearn library and it also deploys combination of decision trees. As evident by the name, it adaptively boosts weak features and scales them up with weights relative to other features ultimately ending up with a combined decision tree classifier to correctly identify the class of a particular feature.

As depicted by the figures below, Adaboost starts by putting a decision boundary and then works upwards by adding weights to weaker features that fall on the wrong side of the decision boundary. Finally once all different iterations are performed, each classifier created is assigned a weight and a final combined decision tree classifier is generated. So based on the same principal of decision trees, Adaboost does not require any feature scaling.



Reference: *Intro to Machine Learning – Lesson 4*

https://www.youtube.com/watch?v=ix6IvwbVpw0

4.4.6 Feature Scaling Deployed or Not

Based on the comparative algorithm testing on the basis of combination of features the algorithm chosen for final analysis was Decision Tree Classifier and as explained in section 4.4.3 it does not require feature selection. So final analysis does not deploy any feature scaling.

5 PICK AND TUNE AN ALGORITHM

NOTE: Section 5 covers different aspects of the algorithm used in the project and addresses questions 3 and 4 of the free-response questionnaire.

Question 3: ***What algorithm did you end up using? What other one(s) did you try?***

5.1 CHOOSING AN ALGORITHM

There were several algorithms tried and tested in order to achieve the required values of the performance metrics, i.e. precision > 0.3 and recall > 0.3.

Algorithm I ended up using in the final project:

- **Decision Tree Classifier**

Other notable algorithms that I used along with the one mentioned above are as follows:

- **Gaussian Naïve Bayes**
- **Support Vector Machine**
- **ADA Boost Classifier**
- **Random Forest Classifier**

NOTE: The decision of choosing the algorithm used in the final analysis was made on the basis of performance matrices generated after evaluating all the algorithms mentioned above. All this is covered in section 7.

Question 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?***

5.2 TUNING THE PARAMETERS OF AN ALGORITHM AND ITS IMPORTANCE

Machine learning is a task to build models from data by self-computation without time consuming human involvement. Part of this includes tuning algorithms, which require machine learners to set certain parameters before their use. The goal of a machine learner is to set the parameters to their optimal values to complete the learning task in the best way possible, i.e. take less computation time and achieve best performance.

Tuning an algorithm in my opinion is a thought process through which a machine learner goes in order to optimize the parameters that impact the model and enable the algorithm to perform the best. Algorithm parameter tuning is an important step for improving algorithm performance metrics such as high accuracy, precision, recall, etc. Parameter tuning if done correctly can save valuable time and cost, for example, achieving high accuracy but not at the cost of lower precision or recall rate.

However, there can be repercussions to all that is mentioned above. Tuning the algorithm, if not done well, may make it more biased towards the training data. In some cases it might be effective but in others this can also lead models to over-fit the testing dataset. This over-fitting can make the algorithm underperform in certain cases.

5.3 TUNING THE ALGORITHMS THAT WERE USED

Getting to the final tuned algorithm was the most time consuming part of the project. I did not get to the algorithm I used in the project right away. I implemented different combinations of features, vectorizer, scaler, feature selector and cross validation techniques to evaluate the Key Performance Indicators (KPIs) i.e. Precision and Recall along with accuracy and F1-score.

Important tools tested in the project from the sklearn library are as follows:

- **GridSearchCV** – *for parameter tuning*
- **TfidfVectorizer** – *for text vectorising*
- **MinMaxScaler** – *for feature scaling*
- **SelectKBest** – *for feature selection*
- **StratifiedShuffleSplit**– *for cross validation*

I have provided the tabular description for all parameters/features settings for the different combinations I tried in order to fulfil the requirements in the project rubric section [Pick an Algorithm and tune an Algorithm]. All of the tuned settings tested are covered in section 7.

6 VALIDATE AND EVALUATE

NOTE: Section 6 will address questions 5 and 6 of the free-response questionnaire and deal with the concepts of validation and evaluation.

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

6.1 VALIDATION

Validation in machine learning is a technique to assess how well the data performs with the model used. The main goal in validation, evident by the name, is to check how the model predicts the testing data set based on the training data set used to train the algorithm, which is measured by a performance metric 'accuracy'.

For this purpose, a common method used is cross-validation. A training and a testing sets are defined and the training data set used to train the model, while the testing data set is used to predict how well it fits the model. Some key advantages of this method are that it gives us the estimate on how will the model perform on an independent data set and also limits over-fitting.

6.2 COMMON MISTAKE WHILE PERFORMING VALIDATION

Validation process starts by splitting the data set into training and testing data sets. The purpose, as mentioned above, is to train the model on the training data set and then predict how well it works with the test data set.

Training the model begins by transforming the training data set using techniques such as, vectorizing (in case of text data), scaling, selection, etc. This is followed by fitting the transformed training data to techniques mentioned above and the predictive algorithm being used (examples shown below).

```
word_features_train = vectorizer.fit_transform(word_features_train).toarray()
```

```
features_train = scaler.fit_transform(features_train)
```

```
clf = GaussianNB()

clf.fit(features_train, labels_train)
pred = clf.predict(features_test)
```

Once the training data is transformed and fitted we move on to the test data set. Here is where the most common mistake might occur. The drill is to make sure not to fit the test data using the scaler, selector, vectorizer and the algorithm itself. This would not give the correct performance metrics, in fact would give better results than expected.

6.3 VALIDATION METHOD USED

Method that was implemented for validation in this project was StratifiedShuffleSplit as implemented in the tester.py program. Imported from sklearn.cross_validation library it is an amalgamation of StratifiedKFold and ShuffleSplit.

Question 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***

6.4 EVALUATION METRICS

The performance of a model is depicted by certain evaluation metrics / performance metrics. Using the bit of code at the end of tester.py program some of these evaluation metrics were determined for each the algorithms tested. Precision and Recall were the primary metrics as a requirement of the project while accuracy of the model was a secondary metric along with the F-scores.

6.5 PERFORMANCE OF THE METRICS

To thoroughly evaluate the average performance of these metrics the folds for cross validation used were set at 1000 so that there were ample re-shuffling & splitting iterations in order to thoroughly average out the performance of each of the evaluation metrics.

6.6 INTERPRETATION OF THE METRICS

Below mentioned is an interpretation of the 2 main evaluation metrics used to evaluate the person of Interest (POI) in the Enron data set. The metric values that we get as an output of the model tell us how good our model is performing, i.e. how correctly is model predicting POI or separating POI from Non-POI. These performance metrics were the deciding factors for the selection of the algorithm for the final analysis.

These metrics can be depicted by contingency matrix below:

		Actual Class	
		POI	*Non-POI*
Predicted Class	*Predicted as POI*	***A -*** *(A given person is in actually a POI & is predicted as a POI*	***B -*** *(A given person is actually a Non-POI & is predicted as a POI*
	Predicted as Non-POI	***C -*** *(A given person is actually a POI & is predicted as a Non- POI*	***D -*** *(A given person is a Non-POI & is predicted as a Non-POI*

A: True Positives **B:** False Positives **C:** False Negatives **D:** True Negatives

RECALL: is the probability of our algorithm to correctly predict a person of interest in the data set, given that the person is actually a person of interest.

RECALL: A / (A+C)

PRECISION: is the ratio of (the actual POIs that were predicted as POIs) to the (total number of persons predicted by the algorithm as POIs).

PRECISION: A / (A+B)

7 TESTING THE ALGORITHMS

7.1 CODE SETUP FOR TESTING THE ALGORITHMS

Code for testing each of the algorithm settings was taken from tester.py program. All the performance testing done is documented in the Ipython notebooks attached as part of the project and the results documented in the tables below.

7.2 NEW FEATURE TESTING ALONG WITH THE ALGORITHM

Sections below explain the decision making process to choose the right algorithm. Part of algorithm performance testing was the creation and testing of new features (2 fraction features and email text features). Details of feature creation/generation were covered in section 4. Performance of different combinations of these features was tested for each of the algorithm used. This is also documented in the Ipython Notebooks mentioned below.

7.3 GENERAL SETTINGS FOR THE ALGORITHMS

The general combination settings used for testing all the algorithms are as below:

Combination	Description
Features selected	All original features(combination) + fraction features + text features
Parameters (SVM)	{'kernel':('linear', 'rbf'), 'C':[1, 10, 100, 1000], 'gamma':[0.1, 0, 1, 10]}
Parameters (DTC)	{'criterion':('gini', 'entropy'), 'max_features':('auto', 'sqrt', 'log2'), 'random_state': [1, 50]}
Parameters (RFC)	{'n_estimators': [1, 250], 'random_state': [1, 50], 'max_features':('auto', 'sqrt', 'log2'), 'criterion':('gini', 'entropy')
Parameters (ADA)	{'n_estimators': [1, 200], 'random_state': [1, 50], 'algorithm' : ['SAMME.R','SAMME']}
Vectorizer	TfidfVectorizer(stop_words="english", lowercase=True) **(Used when email text is being used as a feature)**
Scaler	MinMaxScaler **(Except Decision Tree)**
Selector	SelectKBest(f_classif, k=4)
CV Search	GridSearchCV b
CV Generator	StratifiedShuffleSplit(labels, 1000, random_state = 42) **1000 folds**

7.4 FEATURE TESTING SCENARIOS

In total there were 9 different scenarios of feature combinations tested to evaluate the best performing algorithm. These are as follows:

- **Original Set of Features** *(As list in section 4.1.1)*
- **Selected Set of Features** *(As list in section 4.1.2)*
- **More Selected Set of Features** *(As list in section 4.1.3)*
- **Original Set of Features + Fraction Features** *(2 new features created)*
- **Selected Set of Features + Fraction Features** *(2 new features created)*

- **Original Set of Features + Text Features** *(new email text features created)*
- **Selected Set of Features + Text Features** *(new email text features created)*
- **Original Set of Features + Fraction Features + Text Features**
- **Selected Set of Features + Fraction Features + Text Features**

NOTE: For testing purposes only 500 emails were traversed to collect the text features.

7.5 TEST PERFORMANCE TABLES FOR ALL ALGORITHMS

Using a combination of code from tester.py program and general settings mentioned above, all 9 different scenarios were tested and performance metrics recorded. The resulting performance metrics are tabulated below along with the name of the associated Ipython notebook containing the code that was implemented to a get these results.

7.5.1 Original Set of Features

Algorithm	Precision	Recall	Accuracy	F1	Time (s)
Gaussian NB	0.388	0.294	0.844	0.334	1.059
SVM	0.340	0.036	0.862	0.064	121.757
DT Classifier	0.323	0.347	0.816	0.335	27.927
RF Classifier	0.389	0.202	0.851	0.265	4586.620
ADA Boost	0.296	0.111	0.846	0.162	1355.603

Ipython Notebook: FinalProject_WriteUp_Algo_Test_OriginalFeatures.ipynb

7.5.2 Selected Set of Features

Algorithm	Precision	Recall	Accuracy	F1	Time (s)
Gaussian NB	0.391	0.301	0.844	0.340	0.985
SVM	0.519	0.056	0.867	0.101	121.147
DT Classifier	0.331	0.357	0.818	0.344	27.433
RF Classifier	0.429	0.228	0.857	0.297	4602.665
ADA Boost	0.290	0.107	0.846	0.156	1340.507

Ipython Notebook: FinalProject_WriteUp_Algo_Test_SelectedOriginalFeatures.ipynb

7.5.3 More Selected Set of Features - changes this table

Algorithm	Precision	Recall	Accuracy	F1	Time (s)
Gaussian NB	0.444	0.285	0.835	0.347	0.913
SVM	0.514	0.047	0.847	0.086	119.913
DT Classifier	0.309	0.323	0.785	0.316	27.435
RF Classifier	0.433	0.208	0.836	0.281	4579.593
ADA Boost	0.317	0.093	0.830	0.144	1314.577

Ipython Notebook: FinalProject_WriteUp_Algo_Test_MoreSelectedOriginalFeatures.ipynb

7.5.4 Original Set of Features + Fraction Features

Algorithm	Precision	Recall	Accuracy	F1	Time (s)
Gaussian NB	0.345	0.263	0.835	0.298	1.112
SVM	0.354	0.042	0.862	0.075	125.144
DT Classifier	0.286	0.287	0.809	0.286	29.243
RF Classifier	0.352	0.206	0.844	0.259	2401.868
ADA Boost	0.227	0.074	0.843	0.111	1402.035

Ipython Notebook: FinalProject_WriteUp_Algo_Test_FractionFeatures_OriginalFeatures.ipynb

7.5.5 Selected Set of Features + Fraction Features

Algorithm	Precision	Recall	Accuracy	F1	Time (s)
Gaussian NB	0.349	0.268	0.836	0.303	0.999
SVM	0.477	0.052	0.866	0.094	116.585
DT Classifier	0.296	0.299	0.812	0.297	27.248
RF Classifier	0.380	0.222	0.848	0.280	2319.904
ADA Boost	0.219	0.070	0.843	0.106	1328.101

Ipython Notebook: FinalProject_WriteUp_Algo_Test_FractionFeatures_SelectedOriginalFeatures.ipynb

7.5.6 Original Set of Features + Text Features

Algorithm	Precision	Recall	Accuracy	F1	Time (s)
Gaussian NB	0.310	0.213	0.832	0.252	63.030
SVM	0.335	0.066	0.858	0.110	193.311
DT Classifier	0.289	0.299	0.808	0.294	87.910
RF Classifier	0.348	0.204	0.843	0.257	13717.692
ADA Boost	0.307	0.125	0.846	0.177	1505.744

Ipython Notebook: FinalProject_WriteUp_Algo_Test_TextFeatures_OriginalFeatures.ipynb

7.5.7 Selected Set of Features + Text Features

Algorithm	Precision	Recall	Accuracy	F1	Time (s)
Gaussian NB	0.316	0.216	0.833	0.256	59.188
SVM	0.389	0.075	0.861	0.125	180.579
DT Classifier	0.295	0.304	0.810	0.300	81.257
RF Classifier	0.364	0.211	0.846	0.267	4660.274
ADA Boost	0.316	0.128	0.847	0.182	1419.990

Ipython Notebook: FinalProject_WriteUp_Algo_Test_TextFeatures_SelectedOriginalFeatures.ipynb

7.5.8 Original Set of Features + Fraction Features + Text Features

Algorithm	Precision	Recall	Accuracy	F1	Time (s)
Gaussian NB	0.288	0.202	0.827	0.238	60.436
SVM	0.334	0.064	0.858	0.107	183.437
DT Classifier	0.279	0.285	0.807	0.282	85.019
RF Classifier	0.342	0.188	0.843	0.243	4547.909
ADA Boost	0.242	0.092	0.841	0.133	1444.554

Ipython Notebook: FinalProject_WriteUp_Algo_Test_AllNewFeatures_OriginalFeatures.ipynb

7.5.9 Selected Set of Features + Fraction Features + Text Features

Algorithm	Precision	Recall	Accuracy	F1	Time (s)
Gaussian NB	0.292	0.202	0.828	0.239	57.544
SVM	0.375	0.068	0.861	0.114	172.766
DT Classifier	0.285	0.290	0.808	0.287	78.675
RF Classifier	0.371	0.204	0.848	0.263	4781.990
ADA Boost	0.244	0.092	0.841	0.133	1393.106

Ipython Notebook: FinalProject_WriteUp_Algo_Test_AllNewFeatures_SelectedOriginalFeatures.ipynb

8 ANALYSIS FOR THE FINAL PROJECT

8.1 FINAL ALGORITHM

Based on the results gathered in the tables above for all the different algorithms along with the combination of features, following algorithm was chosen for the final analysis as it seemed to perform better than the other algorithms that were considered.

Decision Tree Classifier

8.2 FINAL FEATURE LIST SELECTION

Details on how the feature list for the final analysis was selected are covered in this section.

NOTE: All the code for various iterations pertaining to final feature set selection is include in ***'FinalProject_WriteUp_Algo_POI_ID_Test.ipynb'***

8.2.1 Original Feature Set

Initially, original feature set was used and the performance metrics of Decision Tree Classifier measured. The settings for running the algorithm were as follows:

Combination	Description
Features selected	All original features (except email address)
Parameters (DTC)	{'criterion':('gini', 'entropy'), 'max_features':('auto', 'sqrt', 'log2'), 'random_state': [1, 50]}
Scaler	MinMax (except Decision Tree)
Selector	SelectKBest(f_classif, k='all')
CV Search	GridSearchCV
CV Generator	StratifiedShuffleSplit(labels, 1000) **1000 folds**

And the performance metrics measured were as follows:

Algorithm	Precision	Recall	Accuracy	F1	Time (s)
DT Classifier	0.285	0.286	0.809	0.285	88.699

8.2.2 Role of Feature Importances

Feature importances are supported by Decision Tree Classifier. Advantage was taken from this fact and 2 very important variables were computed in order to identify the most contributing features towards improving the performance of the algorithm. Following sections explain this in detail. For the following pieces of code, same settings were used as used for documenting the performance metrics of the original feature list. The first variable is documented below.

8.2.3 Average of Feature Importances Score (Original Feature Set)

Average Feature Importances: is defined as the average feature importance score, over 1000 folds, for each feature in the feature list. This exercise was repeated 3 times and scores were recorded to 3 decimal places.

Below is the piece of code which helped compute average feature importances

```
    dt = DecisionTreeClassifier()

    parameters = {'random_state': [1, 50], 'max_features':('auto', 'sqrt', 'log2'), 'criterion':('gini', 'entropy')}
    clf = GridSearchCV(dt, parameters)
    clf.fit(features_train_fit_transformed, labels_train)
    pred = clf.predict(features_test_transformed)

     ### Feature Importance documentation code chunk

    dt.fit(features_train_fit_transformed, labels_train)
    importances = dt.feature_importances_
    important_names_frequency = np_my_feature_list[importances > 0.05]
    important_names_score = np_my_feature_list

    for v in range(len(important_names_score)):

        if important_names_score[v] in feature_importances_score:

            feature_importances_score[important_names_score[v]] += importances[v]
        else:
            feature_importances_score[important_names_score[v]] = 0.0


for key, value in feature_importances_score.items():
    feature_importances_score[key] = round((value / 1000), 3)

print
print 'Average Score of Feature Importances'
pp.pprint(feature_importances_score)
```

This code was run 3 times and average feature importances for 1000 folds recorded in a table. Based on the importance score I identified few feature as possible candidates for the final feature list. The table is as follows:

Original Features	Average Feature Importances			*Average Scores*	*Frequency Score Ranks by Intuition*
	Iteration 1	*Iteration 2*	*Iteration 3*		
salary	0.019	0.018	0.018	0.018	*Not Considered*
deferral_payments	0.007	0.008	0.007	0.007	*Not Considered*
total_payments	0.032	0.034	0.035	0.034	*Not Considered*
loan_advances	0.001	0.001	0.001	0.001	*Not Considered*
bonus	0.089	0.086	0.087	0.087	3
restricted_stock_deferred	0.001	0.001	0.001	0.001	*Not Considered*
deferred_income	0.035	0.035	0.035	0.035	*Not Considered*
total_stock_value	0.058	0.057	0.056	0.057	3
expenses	0.107	0.108	0.108	0.108	2
exercised_stock_options	0.144	0.145	0.140	0.143	1
other	0.119	0.113	0.121	0.118	2
long_term_incentive	0.030	0.029	0.029	0.029	*Not Considered*
restricted_stock	0.037	0.036	0.038	0.037	*Not Considered*
director_fees	0.000	0.000	0.000	0.000	*Not Considered*
to_messages	0.009	0.009	0.009	0.009	*Not Considered*
from_poi_to_this_person	0.011	0.011	0.010	0.011	*Not Considered*
from_messages	0.017	0.018	0.016	0.017	*Not Considered*
from_this_person_to_poi	0.019	0.020	0.019	0.019	*Not Considered*
shared_receipt_with_poi	0.110	0.113	0.112	0.112	2
fraction_from_poi	0.008	0.010	0.009	0.009	*Not Considered*
fraction_to_poi	0.147	0.148	0.148	0.148	1

I took further averages of the importances recorded for each of the features and ranked them intuitively for future reference when creating final feature list. Rank '1' was given to the features with average importance greater than 0.4, rank '2' was assigned to the features with importance less than 0.4 and greater than 0.1. Finally, rank '3' was given to features with scores less than 0.1 and 0.05.

The other variable based on feature importances is documented below.

8.2.4 Frequency of Feature Importances (Original Feature Set)

Feature Importances Frequency: is defined as the number of times a particular feature appears as an important feature from a total of 1000 iterations. For a feature to be considered as an important feature, it should have importance score of 0.05 or more. Following piece of code was used to compute frequencies of these features.

```python
    dt = DecisionTreeClassifier()

    parameters = {'random_state': [1, 50], 'max_features':('auto', 'sqrt', 'log2'), 'criterion':('gini', 'entropy')}
    clf = GridSearchCV(dt, parameters)
    clf.fit(features_train_fit_transformed, labels_train)
    pred = clf.predict(features_test_transformed)

     ### Feature Importance documentation code chunk

    dt.fit(features_train_fit_transformed, labels_train)
    importances = dt.feature_importances_
    important_names_frequency = np_my_feature_list[importances > 0.05]
    important_names_score = np_my_feature_list


    for v in range(len(important_names_frequency)):

        if important_names_frequency[v] in feature_importances_frequency:

            feature_importances_frequency[important_names_frequency[v]] += 1
        else:
            feature_importances_frequency[important_names_frequency[v]] = 1
print 'Frequency of Feature Importances'
print
pp.pprint(feature_importances_frequency)
```

Again this code was also run 3 times and frequency/occurrence of important features for 1000 folds was recorded in a table. Based on the occurrence of the feature out of the 1000 times algorithm was run I identified another set of possible candidates for the final feature list. These occurrences per 1000 folds are documented below:

Original Features	**Feature Importances Frequency**			*Average Frequency*	*Feature Frequency Ranks by Intuition*
	Iteration 1	*Iteration 2*	*Iteration 3*		
salary	140	137	154	143.667	3
deferral_payments	58	69	54	60.333	*Not Considered*
total_payments	264	281	265	270.000	2
loan_advances	12	13	23	16.000	*Not Considered*
bonus	392	397	413	400.667	1
restricted_stock_deferred	10	6	8	8.000	*Not Considered*
deferred_income	242	255	242	246.333	*Not Considered*
total_stock_value	560	557	547	554.667	1
expenses	712	731	733	725.333	1
exercised_stock_options	760	767	766	764.333	1
other	770	745	772	762.333	1
long_term_incentive	248	227	241	238.667	2
restricted_stock	272	248	285	268.333	2
director_fees	--	--	--	--	*Not Considered*
to_messages	87	86	96	89.667	*Not Considered*
from_poi_to_this_person	89	87	88	88.000	*Not Considered*
from_messages	155	159	133	149.000	3
from_this_person_to_poi	205	207	209	207.000	2
shared_receipt_with_poi	666	669	668	667.667	1
fraction_from_poi	71	85	73	76.333	*Not Considered*
fraction_to_poi	903	893	902	899.333	1

Here I also took average frequency/occurrence of the important features and ranked them intuitively for future reference when creating final feature list. Rank ‘1’ was given to the features that occurred more than 400 times, out of the total 1000, in the important feature list. Subsequently, rank ‘2’ was assigned to the features which occurred less than 400 times and more than 200 times as important features. Rank ‘3’ was given to features with average occurrences less than 200.

All of the computation made above to rank the feature laid a benchmark and made it very easy for me to create the final feature list.

8.2.5 Filtered Feature Set

This filtered feature set was created based on the features importances documented in the tables above. Now the table below shows selection of feature based on the intersection of the 2 variables documented above.

Original Features	*Frequency Score Ranks by Intuition*	*Feature Frequency Ranks by Intuition*	*Filterd Feature Selection by Intuition*
salary	*Not Considered*	3	**No**
deferral_payments	*Not Considered*	*Not Considered*	**No**
total_payments	*Not Considered*	2	**No**
loan_advances	*Not Considered*	*Not Considered*	**No**
bonus	3	1	**Yes**
restricted_stock_deferred	*Not Considered*	*Not Considered*	**No**
deferred_income	*Not Considered*	*Not Considered*	**No**
total_stock_value	3	1	**Yes**
expenses	2	1	**Yes**
exercised_stock_options	1	1	**Yes**
other	2	1	**Yes**
long_term_incentive	*Not Considered*	2	**No**
restricted_stock	*Not Considered*	2	**No**
director_fees	*Not Considered*	*Not Considered*	**No**
to_messages	*Not Considered*	*Not Considered*	**No**
from_poi_to_this_person	*Not Considered*	*Not Considered*	**No**
from_messages	*Not Considered*	3	**No**
from_this_person_to_poi	*Not Considered*	2	**No**
shared_receipt_with_poi	2	1	**Yes**
fraction_from_poi	*Not Considered*	*Not Considered*	**No**
fraction_to_poi	1	1	**Yes**

The filtered feature list was created by including all the features marked as ‘Yes’. This filtered feature list was then used as an input for the decision tree algorithm. The settings for running the algorithm were same as before except the features list:

Combination	**Description**
Features selected	Filtered Feature List
Parameters (DTC)	{'criterion':('gini', 'entropy'), 'max_features':('auto', 'sqrt', 'log2'), 'random_state': [1, 50]}
Scaler	MinMax (except Decision Tree)
Selector	SelectKBest(f_classif, k='all')
CV Search	GridSearchCV
CV Generator	StratifiedShuffleSplit(labels, 1000) **1000 folds**

The performance metrics measured with the filtered feature set were as follows:

Algorithm	Precision	Recall	Accuracy	F1	Time (s)
DT Classifier	0.465	0.427	0.848	0.445	92.108

Below mentioned is the filtered feature list used:

```
['poi',
 'bonus',
 'total_stock_value',
 'expenses',
 'exercised_stock_options',
 'other',
 'shared_receipt_with_poi',
 'fraction_to_poi']
```

8.2.6 Average of Feature Importances Score and Frequency (Filtered Feature Set)

3 iterations were run for the algorithm with the filtered feature set, average feature importance and frequencies were documented in the tables below:

Average of Feature Importances Score

Original Features	Average Feature Importances			*Average Scores*
	Iteration 1	*Iteration 2*	*Iteration 3*	
bonus	0.106	0.106	0.103	0.105
total_stock_value	0.059	0.060	0.060	0.060
expenses	0.163	0.158	0.164	0.162
exercised_stock_options	0.154	0.149	0.146	0.150
other	0.185	0.186	0.188	0.186
shared_receipt_with_poi	0.157	0.161	0.156	0.158
fraction_to_poi	0.176	0.178	0.182	0.179

Feature Importances Frequency

Original Features	Feature Importances Frequency			*Average Frequency*
	Iteration 1	*Iteration 2*	*Iteration 3*	
bonus	504	505	476	495.000
total_stock_value	485	498	486	489.667
expenses	872	864	882	872.667
exercised_stock_options	789	787	763	779.667
other	938	942	942	940.667
shared_receipt_with_poi	851	870	851	857.333
fraction_to_poi	985	989	996	990.000

The tables above show the improvement in the both the average score and frequency. Feature 'total_stock_value' seems to be the weakest of them all followed by 'bonus' and 'exercised_stock_options'.

Taking this analysis further, I had 2 options for the implementation of the algorithm.

1. Creating a subset of this filtered feature set excluding 'total_stock_value', 'bonus' and 'exercised_stock_options'.
2. Using k = 4 in SelectKBest for the best possible selection of features.

Performance metrics with both the options were tabulated and best one selected for the final implementation of the algorithm.

Option 1

Algorithm	Precision	Recall	Accuracy	F1	Time (s)
DT Classifier	0.402	0.377	0.831	0.389	85.478

Option 2

Algorithm	Precision	Recall	Accuracy	F1	Time (s)
DT Classifier	0.465	0.427	0.848	0.445	84.255

Clearly the favorite was option 2

8.3 FINAL FEATURE LIST

Feature list chosen for the final implementation of the algorithm was same as the filtered feature list/set. 'poi' plus 6 other best performing features were selected from the original feature list while an additional feature 'fraction_to_poi' was selected from the 2 newly created features.

```
features_list = ['poi', 'bonus', 'total_stock_value', 'expenses',
                 'exercised_stock_options', 'other','shared_receipt_with_poi']
```

+

```
['fraction_to_poi']
```

```
['poi',
 'bonus',
 'total_stock_value',
 'expenses',
 'exercised_stock_options',
 'other',
 'shared_receipt_with_poi',
 'fraction_to_poi']
```

9 IMPLEMENTING THE FINAL ALGORITHM

For the implementation of the algorithm, poi_id.py program was modified and executed in order to evaluate the performance. Following are the code chunk snap shots taken from the poi_id.py program.

9.1 IMPORT STATEMENTS, FEATURES AND LOADING THE DATA

```
#!/usr/bin/python

import sys
import pickle
import os
import re
import string
import numpy as np
import pprint as pp
sys.path.append("../tools/")

### for measuring time taken

import time

from feature_format import featureFormat, targetFeatureSplit
from tester import test_classifier, dump_classifier_and_data

### Task 1: Select what features you'll use.
### features_list is a list of strings, each of which is a feature name.
### The first feature must be "poi".

print
print 'Features originally chosen:'

features_list = ['poi', 'bonus', 'total_stock_value', 'expenses',
                 'exercised_stock_options', 'other','shared_receipt_with_poi'] # You will need to use more features

print
pp.pprint(features_list)

time.sleep(5)

### Load the dictionary containing the dataset
data_dict = pickle.load(open("final_project_dataset.pkl", "r") )
```

9.2 REMOVING OUTLIERS

```
### Task 2: Remove outliers

print
print 'Outliers removed:'
print
print '[GRAMM WENDY L]'
print
data_dict.pop("GRAMM WENDY L", None)
print '[LOCKHART EUGENE E]'
print
data_dict.pop("LOCKHART EUGENE E", None)
print '[WROBEL BRUCE]'
print
data_dict.pop("WROBEL BRUCE", None)
print '[THE TRAVEL AGENCY IN THE PARK]'
print
data_dict.pop("THE TRAVEL AGENCY IN THE PARK", None)
print '[TOTAL]'
data_dict.pop("TOTAL", None)
```

9.3 UPDATING INCONSISTENCIES

```
print
print 'Inconsistent records updated:'
print
print '[BELFER ROBERT]'
print
print '[BHATNAGAR SANJAY]'


data_dict['BELFER ROBERT'] = {'bonus': 'NaN',
                              'deferral_payments': 'NaN',
                              'deferred_income': -102500,
                              'director_fees': 102500,
                              'email_address': 'NaN',
                              'exercised_stock_options': 'NaN',
                              'expenses': 3285,
                              'from_messages': 'NaN',
                              'from_poi_to_this_person': 'NaN',
                              'from_this_person_to_poi': 'NaN',
                              'loan_advances': 'NaN',
                              'long_term_incentive': 'NaN',
                              'other': 'NaN',
                              'poi': False,
                              'restricted_stock': -44093,
                              'restricted_stock_deferred': 44093,
                              'salary': 'NaN',
                              'shared_receipt_with_poi': 'NaN',
                              'to_messages': 'NaN',
                              'total_payments': 3285,
                              'total_stock_value': 'NaN'}

data_dict['BHATNAGAR SANJAY'] = {'bonus': 'NaN',
                                 'deferral_payments': 'NaN',
                                 'deferred_income': 'NaN',
                                 'director_fees': 'NaN',
                                 'email_address': 'sanjay.bhatnagar@enron.com',
                                 'exercised_stock_options': 15456290,
                                 'expenses': 137864,
                                 'from_messages': 29,
                                 'from_poi_to_this_person': 0,
                                 'from_this_person_to_poi': 1,
                                 'loan_advances': 'NaN',
                                 'long_term_incentive': 'NaN',
                                 'other': 'NaN',
                                 'poi': False,
                                 'restricted_stock': 2604490,
                                 'restricted_stock_deferred': -2604490,
                                 'salary': 'NaN',
                                 'shared_receipt_with_poi': 463,
                                 'to_messages': 523,
                                 'total_payments': 137864,
                                 'total_stock_value': 15456290}
```

9.4 New Feature Creation

```
### Task 3: Create new feature(s)

print
print 'Creating 1 new feature that is to be used as part of the final analysis'


### Store to my_dataset for easy export below.

my_dataset = data_dict
my_feature_list = features_list

np_my_feature_list = np.array(my_feature_list)

### computeFraction function used from lesson 11 exercise

def computeFraction(poi_messages, all_messages):

    if (poi_messages == 'NaN') or (all_messages == 'NaN'):
        fraction = 0.
    else:
        fraction = float(poi_messages)/float(all_messages)

    return fraction


for name in data_dict:

    data_point = data_dict[name]
    from_this_person_to_poi = data_point["from_this_person_to_poi"]
    from_messages = data_point["from_messages"]
    fraction_to_poi = computeFraction(from_this_person_to_poi, from_messages)


    ### Adding values of new features to the modified dataset, 'my_dataset'

    my_dataset[name]['fraction_to_poi'] = fraction_to_poi

time.sleep(5)

print
print '1 new feature, "fraction_to_poi", added to "my_dataset"'


### Adding new feature to the modified features list, 'my_feature_list'

fraction_features = ['fraction_to_poi']

my_feature_list = features_list + fraction_features

np_my_feature_list = np.array(my_feature_list)

np_my_feature_list = np_my_feature_list[1::]

print
print 'Number of features now:', len(my_feature_list), "and the final features list:"
print
pp.pprint(my_feature_list)
```

9.5 ADDITIONAL IMPORT STATEMENTS AND INITIATIONS

```
### Extract features and labels from dataset for local testing

data = featureFormat(my_dataset, my_feature_list, sort_keys = True)
labels, features = targetFeatureSplit(data)

### Task 4: Try a varity of classifiers

### Feature Selector utility

from sklearn.feature_selection import SelectKBest, f_classif

### Grid Fit/Transform utility

from sklearn.grid_search import GridSearchCV

### Algorithm

from sklearn.tree import DecisionTreeClassifier

### Cross Validation utility

from sklearn.cross_validation import StratifiedShuffleSplit


### Please name your classifier clf for easy export below.
### Note that if you want to do PCA or other multi-stage operations,
### you'll need to use Pipelines. For more info:
### http://scikit-learn.org/stable/modules/pipeline.html

### from sklearn.naive_bayes import GaussianNB
### clf = GaussianNB()    # Provided to give you a starting point. Try a varity of classifiers.

folds = 1000
cv = StratifiedShuffleSplit(labels, folds)


### feature selection, because text is super high dimensional and
### can be really computationally chewy as a result

k=4
selector = SelectKBest(f_classif, k)


### counter for number of iterations

count = 0

### dictionaries to document feature importances

feature_importances_frequency = {}
feature_importances_score = {}
```

9.6 Training the Algorithm and Feature Importances

```
for train_indices, test_indices in cv:
    features_train = []
    features_test = []
    word_features_train = []
    word_features_test = []
    labels_train = []
    labels_test = []

    ### Partitioning of the data into training and testing datasets

    features_train = [features[ii] for ii in train_indices]
    labels_train = [labels[ii] for ii in train_indices]

    features_test = [features[jj] for jj in test_indices]
    labels_test = [labels[jj] for jj in test_indices]

    ### Fitting and transformation of the training dataset (vectorizer, scaling, feature selection)

    features_train_fit_transformed = selector.fit_transform(features_train, labels_train)

    ### Transformation of the test dataset (vectorizer, scaling, feature selection)

    features_test_transformed = selector.transform(features_test)

    ### Training the classifer

    ### Decision Tree Classifier code chunk

    dt = DecisionTreeClassifier()

    parameters = {'random_state': [1, 50], 'max_features':('auto', 'sqrt', 'log2'), 'criterion':('gini', 'entropy')}
    clf = GridSearchCV(dt, parameters)
    clf.fit(features_train_fit_transformed, labels_train)
    pred = clf.predict(features_test_transformed)

     ### Feature Importance documentation code chunk

    dt.fit(features_train_fit_transformed, labels_train)
    importances = dt.feature_importances_
    important_names_frequency = np_my_feature_list[importances > 0.05]
    important_names_score = np_my_feature_list

    count += 1
    if count%10 == 0:
        print '*',
    if count%100 == 0:
        print count/10,'%'


    for v in range(len(important_names_frequency)):

        if important_names_frequency[v] in feature_importances_frequency:

            feature_importances_frequency[important_names_frequency[v]] += 1
        else:
            feature_importances_frequency[important_names_frequency[v]] = 1

    for v in range(4):

        if important_names_score[v] in feature_importances_score:

            feature_importances_score[important_names_score[v]] += importances[v]
        else:
            feature_importances_score[important_names_score[v]] = 0.0


for key, value in feature_importances_score.items():
    feature_importances_score[key] = round((value / 1000), 3)

print
print 'Model Trained'
print
print "Total time taken to train:", round(time.time()-t0, 3), "s"
```

9.7 PERFORMANCE METRICS OF THE FINAL MODEL

Algorithm	Precision	Recall	Accuracy	F1	Time (s)
	0.465	0.427	0.848	0.445	92.108
DT Classifier	Predictions	True Positives	False Positives	False Negatives	True Negatives
	14000	853	983	1147	11017

9.8 FEATURES IMPORTANCES OF THE BEST PERFORMING FEATURES

Decision Tree Classifiers supports feature importances. As feature selection as implemented using SelectKBest with k=4, feature importances were documented.

Original Features	*Feature Importances Frequency*	*Average Feature Importances*
bonus	996	0.26
total_stock_value	904	0.16
expenses	987	0.289
exercised_stock_options	1000	0.29

10 CONCLUSION

Overall this project was an amalgamation of a lot of aspects of machine learning. The data set, although, quite small presented interesting challenges. To name a few:

- **Selecting the appropriate features**
- **Identifying the outliers**
- **Updating some inconsistent observations**
- **Creating new features for improvement in performance**
- **Implementing feature selection**

Some of the machine learning techniques mentioned above were engaging and fun to code.

11 References

http://stackoverflow.com/questions/3545331/how-can-i-get-dictionary-key-as-variable-directly-in-python-not-by-searching-fr

http://stackoverflow.com/questions/18448847/python-import-txt-file-and-having-each-line-as-a-list

http://stackoverflow.com/questions/16864941/python-count-items-in-dict-value-that-is-a-list

http://stackoverflow.com/questions/13670945/formatting-last-name-first-name

http://stackoverflow.com/questions/6797984/how-to-convert-string-to-lowercase-in-python

http://stackoverflow.com/questions/3182183/creating-a-list-of-objects-in-python

http://www.tutorialspoint.com/python/list_append.htm

http://texthandler.com/?module=remove_line_breaks_python

http://stackoverflow.com/questions/11697709/comparing-two-lists-in-python

http://stackoverflow.com/questions/7313157/python-create-list-of-tuples-from-lists

https://docs.python.org/2/library/functions.html#max

https://docs.python.org/2/library/functions.html#abs

http://stackoverflow.com/questions/16548668/iterating-over-a-2-dimensional-python-list

http://anh.cs.luc.edu/python/hands-on/3.1/handsonHtml/loops.html

http://stackoverflow.com/questions/13039192/max-second-element-in-tuples-python

https://docs.python.org/3.5/library/functions.html#zip

http://stackoverflow.com/questions/3308102/how-to-extract-the-n-th-elements-from-a-list-of-tuples-in-python

http://stackoverflow.com/questions/5656670/remove-max-and-min-values-from-python-list-of-integers

http://stackoverflow.com/questions/11277432/how-to-remove-a-key-from-a-dictionary

http://stackoverflow.com/questions/21593689/dividing-by-zero-error-in-python

http://stackoverflow.com/questions/6726636/splitting-and-concatenating-a-string

http://stackoverflow.com/questions/23792781/tf-idf-feature-weights-using-sklearn-feature-extraction-text-tfidfvectorizer

http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html#examples-using-sklearn-feature-extraction-text-tfidfvectorizer

http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

http://stackoverflow.com/questions/2474015/getting-the-index-of-the-returned-max-or-min-item-using-max-min-on-a-list

http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

http://stackoverflow.com/questions/4056768/how-to-declare-array-of-zeros-in-python-or-an-array-of-a-certain-size

http://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html

http://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html

http://stackoverflow.com/questions/19410042/how-to-make-ipython-notebook-inline-matplotlib-graphics http://nbviewer.ipython.org/github/UoLPythonGroup/training/blob/master/matplotlib/matplotlib%20examples.ipynb

http://www.tutorialspoint.com/python/dictionary_keys.htm

https://docs.python.org/2/library/stdtypes.html#dict.itervalues

http://en.wikipedia.org/wiki/Enron

http://en.wikipedia.org/wiki/Enron_scandal

http://scikit-learn.org/stable/modules/feature_selection.html

https://docs.python.org/2/library/os.html

https://docs.python.org/2/library/os.html#os.listdir

https://www.youtube.com/watch?v=BVbUM2DVykc

http://comments.gmane.org/gmane.comp.python.scikit-learn/11392

http://discussions.udacity.com/t/final-project-feature-selection/4621 (specifically for using tfidf for text vectorizer)

https://www.youtube.com/watch?v=4Li5-AU7-yA

http://stackoverflow.com/questions/21892570/ipython-notebook-align-table-to-the-left-of-cell

http://ipython.org/ipython-doc/2/api/generated/IPython.core.display.html#IPython.core.display.DisplayObject

http://discussions.udacity.com/t/two-records-financial-values-out-of-sync/8687

http://stackoverflow.com/questions/22903267/what-is-tuning-in-machine-learning

http://machinelearningmastery.com/how-to-improve-machine-learning-results/

http://discussions.udacity.com/t/final-project-feature-selection/4621/3

http://scikit-learn.org/0.12/auto_examples/ensemble/plot_forest_importances.html

http://stackoverflow.com/questions/16341631/counting-same-elements-in-an-array-and-create-dictionary

http://en.wikipedia.org/wiki/Cross-validation_%28statistics%29

http://en.wikipedia.org/wiki/Precision_and_recall

http://stackoverflow.com/questions/22361781/how-does-sklearn-random-forest-index-feature-importances

https://www.youtube.com/watch?v=2akd6uwtowc

http://stackoverflow.com/questions/6632188/explicitly-select-items-from-a-python-list-or-tuple

http://stackoverflow.com/questions/15536623/modify-dict-values-inplace