

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”]

Enron Corporation was U.S. energy-trading and utilities company founded in 1985. It achieved spectacular success in a very short period of time. This company claimed revenues of around 101 billion dollars in 2000. In only 15 years, this company was named America's Most innovative Company for six consecutive years by Fortune 500. Company stock price rose to 90 dollars. However, this success of Enron was short-lived. It started falling off more rapidly and bankruptcy was filed on Dec.2, 2001. Federal Energy Regulatory Commission investigated after the company's collapse and approximately 600,000 emails generated by employees of the Enron Corporation was made public which is called enron corpus.

In this project, main goal is to build a classification algorithm to predict a person of interest identifier (POI) based on email and financial features in the enron dataset involving 146 enron executives. A POI is anyone who is indicted, settled without admitting the guilt and testified in exchange for immunity. Predicted POI from this analysis and actual POI in the dataset will be compared to validate my prediction. Such model could be used to find additional suspects who were not indicted during the original investigation, or to find persons of interest during fraud investigations at other businesses. This study involves:

-Exploration of Enron dataset

-Selection of Features

-Selection of Algorithms

-Validation and Evaluation

```
In [64]: import matplotlib.pyplot as plt
        from IPython.display import Image
        import numpy as np
        import pandas as pd
        import sys
        import pickle
        from sklearn.naive_bayes import GaussianNB
        from sklearn.metrics import accuracy_score
        from sklearn.metrics import precision_score
        from sklearn.metrics import recall_score
        from sklearn.metrics import recall_score, precision_score
        from time import time

        sys.path.append("../tools/")

        from feature_format import featureFormat, targetFeatureSplit
```

```
In [65]: with open("final_project_dataset.pkl", "r") as data_file:
         data_dict = pickle.load(data_file)
```

```
In [66]: enron_data = pd.DataFrame.from_dict(data_dict, orient = 'index')
```

Exploration of Enron Dataset

Dataset was investigated. First structure of the dataset was examined.

```
In [67]: enron_data.head()
```

Out[67]:

	salary	to_messages	deferral_payments	total_payments	exercised_s
ALLEN PHILLIP K	201955	2902	2869717	4484442	1729541
BADUM JAMES P	NaN	NaN	178980	182466	257817
BANNANTINE JAMES M	477	566	NaN	916197	4046157
BAXTER JOHN C	267102	NaN	1295738	5634343	6680544
BAY FRANKLIN R	239671	NaN	260455	827696	NaN

5 rows × 21 columns

```
In [68]: #structure of enron_data set
         print "There were", len(enron_data), "persons involved."
```

There were 146 persons involved.

```
In [69]: print data_dict.keys()
```

['METTS MARK', 'BAXTER JOHN C', 'ELLIOTT STEVEN', 'CORDES WILLIAM R',
, 'HANNON KEVIN P', 'MORDAUNT KRISTINA M', 'MEYER ROCKFORD G', 'MCMAHON JEFFREY', 'HORTON STANLEY C', 'PIPER GREGORY F', 'HUMPHREY GENE E', 'UMANOFF ADAM S', 'BLACHMAN JEREMY M', 'SUNDE MARTIN', 'GIBBS DANA R', 'LOWRY CHARLES P', 'COLWELL WESLEY', 'MULLER MARK S', 'JACKSON CHARLENE R', 'WESTFAHL RICHARD K', 'WALTERS GARETH W', 'WALLS JR ROBERT H', 'KITCHEN LOUISE', 'CHAN RONNIE', 'BELFER ROBERT', 'SHANKMAN JEFFREY A', 'WODRASKA JOHN', 'BERGSIEKER RICHARD P', 'URQUHART JOHN A', 'BIBI PHILIPPE A', 'RIEKER PAULA H', 'WHALEY DAVID A', 'BECK SALLY W', 'HAUG DAVID L', 'ECHOLS JOHN B', 'MENDELSON JOHN', 'HICKERSON GARY J', 'CLINE KENNETH W', 'LEWIS RICHARD', 'HAYES ROBERT E', 'MCCARTY DANNY J', 'KOPPER MICHAEL J', 'LEFF DANIEL P', 'LAVORATO JOHN J', 'BERBERIAN DAVID', 'DETMERING TIMOTHY J', 'WAKEHAM JOHN', 'POWERS WILLIAM', 'GOLD JOSEPH', 'BANNANTINE JAMES M', 'DUNCAN JOHN H', 'SHAPIRO RICHARD S', 'SHERRIFF JOHN R', 'SHELBY REX', 'LEMAISTRE CHARLES', 'DEFFNER JOSEPH M', 'KISHKILL JOSEPH G', 'WHALLEY LAWRENCE G', 'MCCONNELL MICHAEL S', 'PIRO JIM', 'DELAINEY DAVID W', 'SULLIVAN-SHAKLOVITZ COLLEEN', 'WROBEL BRUCE', 'LINDHOLM TOD A', 'MEYER JEROME J', 'LAY KENNETH L', 'BUTTS ROBERT H', 'OLSON CINDY K', 'MCDONALD REBECCA', 'CUMBERLAND MICHAEL S', 'GAHN ROBERT S', 'MCCLELLAN GEORGE', 'HERMANN ROBERT J', 'SCRIMSHAW MATTHEW', 'GATHMANN WILLIAM D', 'HAEDICKE MARK E', 'BOWEN JR RAYMOND M', 'GILLIS JOHN', 'FITZGERALD JAY L', 'MORAN MICHAEL P', 'REDMOND BRIAN L', 'BAZELIDES PHILIP J', 'BELDEN TIMOTHY N', 'DURAN WILLIAM D', 'THORN TERENCE H', 'FASTOW ANDREW S', 'FOY JOE', 'CALGER CHRISTOPHER F', 'RICE KENNETH D', 'KAMINSKI WINCENTY J', 'LOCKHART EUGENE E', 'COX DAVID', 'OVERDYKE JR JERE C', 'PEREIRA PAULO V. FERRAZ', 'STABLER FRANK', 'SKILLING JEFFREY K', 'BLAKE JR. NORMAN P', 'SHERRICK JEFFREY B', 'PRENTICE JAMES', 'GRAY RODNEY', 'PICKERING MARK R', 'THE TRAVEL AGENCY IN THE PARK', 'NOLES JAMES L', 'KEAN STEVEN J', 'TOTAL', 'FOWLER PEGGY', 'WASAFF GEORGE', 'WHITE JR THOMAS E', 'CHRISTODOULOU DIOMEDES', 'ALLEN PHILLIP K', 'SHARP VICTORIA T', 'JAEDICKE ROBERT', 'WINOKUR JR. HERBERT S', 'BROWN MICHAEL', 'BADUM JAMES P', 'HUGHES JAMES A', 'REYNOLDS LAWRENCE', 'DIMICHELE RICHARD G', 'BHATNAGAR SANJAY', 'CARTER REBECCA C', 'BUCHANAN HAROLD G', 'YEAP SOON', 'MURRAY JULIA H', 'GARLAND C KEVIN', 'DODSON KEITH', 'YEAGER F SCOTT', 'HIRKO JOSEPH', 'DIETRICH JANET R', 'DERRICK JR. JAMES V', 'FREVERT MARK A', 'PAI LOU L', 'BAY FRANKLIN R', 'HAYSLETT RODERICK J', 'FUGH JOHN L', 'FALLON JAMES B', 'KOENIG MARK E', 'SAVAGE FRANK', 'IZZO LAWRENCE L', 'TILNEY ELIZABETH A', 'MARTIN AMANDA K', 'BUY RICHARD B', 'GRAMM WENDY L', 'CAUSEY RICHARD A', 'TAYLOR MITCHELL S', 'DONAHUE JR JEFFREY M', 'GLISAN JR BEN F']

```
In [70]: print data_dict["GLISAN JR BEN F"]
```

```
{'salary': 274975, 'to_messages': 873, 'deferral_payments': 'NaN', 'total_payments': 1272284, 'exercised_stock_options': 384728, 'bonus': 600000, 'restricted_stock': 393818, 'shared_receipt_with_poi': 874, 'restricted_stock_deferred': 'NaN', 'total_stock_value': 778546, 'expenses': 125978, 'loan_advances': 'NaN', 'from_messages': 16, 'other': 200308, 'from_this_person_to_poi': 6, 'poi': True, 'director_fees': 'NaN', 'deferred_income': 'NaN', 'long_term_incentive': 71023, 'email_address': 'ben.glisan@enron.com', 'from_poi_to_this_person': 52}
```

```
In [71]: (enron_data['poi'].value_counts()[True])
```

```
Out[71]: 18
```

```
In [72]: enron_data.describe().transpose()
```

Out[72]:

	count	unique	top	freq
salary	146	95	NaN	51
to_messages	146	87	NaN	60
deferral_payments	146	40	NaN	107
total_payments	146	126	NaN	21
exercised_stock_options	146	102	NaN	44
bonus	146	42	NaN	64
restricted_stock	146	98	NaN	36
shared_receipt_with_poi	146	84	NaN	60
restricted_stock_deferred	146	19	NaN	128
total_stock_value	146	125	NaN	20
expenses	146	95	NaN	51
loan_advances	146	5	NaN	142
from_messages	146	65	NaN	60
other	146	93	NaN	53
from_this_person_to_poi	146	42	NaN	60
poi	146	2	False	128
director_fees	146	18	NaN	129
deferred_income	146	45	NaN	97
long_term_incentive	146	53	NaN	80
email_address	146	112	NaN	35
from_poi_to_this_person	146	58	NaN	60

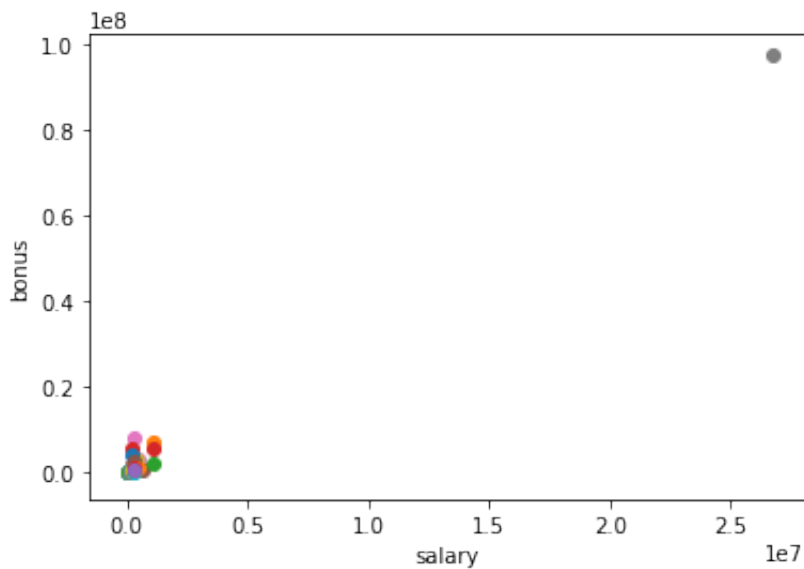
Enron dataset consists of lot of missing values (NaN). All the features have at least one missing values. Some features have more than 50% of their values missing, as shown by the table above. NaNs are replaced by 0 for training our algorithm later.

```
In [73]: enron_data.replace(to_replace='NaN', value=0.0, inplace=True)
```

Scatter plots are great tools for finding outliers. So first I plotted salary against bonus in a scatter plot and looked for the data distribution.

```
In [74]: from IPython.display import Image
features = ["salary", "bonus"]
#data_dict.pop('TOTAL', 0)
data = featureFormat(data_dict, features)
### plot features
for point in data:
    salary = point[0]
    bonus = point[1]
    plt.scatter( salary, bonus )

plt.xlabel("salary")
plt.ylabel("bonus")
plt.show()
```



The above plot allowed to see an outlier named 'Total' which was sum of all the data points in the given plot (artifact in scatterplot). So this outlier was excluded from our analysis manually. There were several other high number values (outliers) which could possibly represent POIs, thus were included in the dataset. Scatter plot was drawn after removing outliers as below.

```
In [75]: ##2 removing outliers
features = ["salary", "bonus"]
data_dict.pop('TOTAL', 0)
data = featureFormat(data_dict, features)

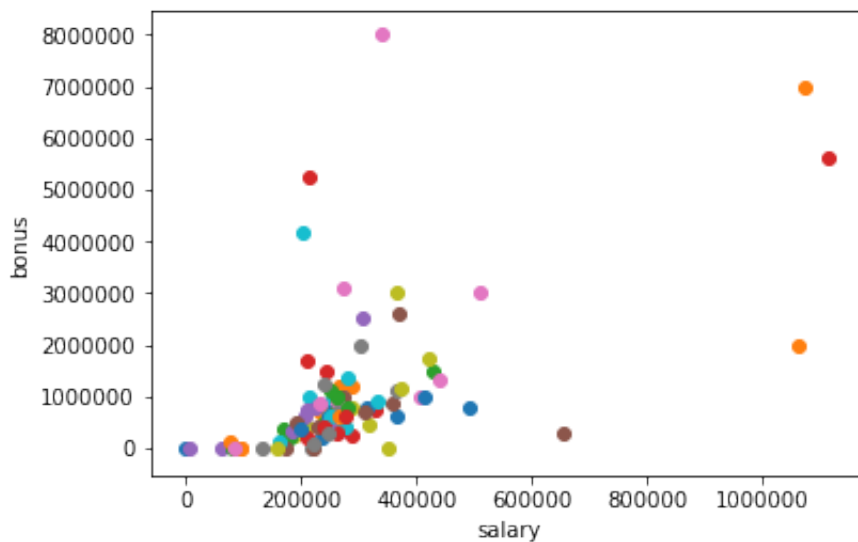
#remove NAN from dataset
outliers = []
for key in data_dict:
    val = data_dict[key]['salary']
    if val == 'NaN':
        continue
    outliers.append((key, int(val)))
outliers_final = (sorted(outliers, key=lambda x:x[1], reverse = True)[:10])
print outliers_final

[('SKILLING JEFFREY K', 1111258), ('LAY KENNETH L', 1072321), ('FREV
ERT MARK A', 1060932), ('PICKERING MARK R', 655037), ('WHALLEY LAWRE
NCE G', 510364), ('DERRICK JR. JAMES V', 492375), ('FASTOW ANDREW S'
, 440698), ('SHERRIFF JOHN R', 428780), ('RICE KENNETH D', 420636),
('CAUSEY RICHARD A', 415189)]
```

```
In [76]: #scatterplot after outlier removal

features = ["salary", "bonus"]
#data_dict.pop('TOTAL', 0)
data = featureFormat(data_dict, features)
### plot features
for point in data:
    salary = point[0]
    bonus = point[1]
    plt.scatter( salary, bonus )

plt.xlabel("salary")
plt.ylabel("bonus")
plt.show()
```



Feature engineering

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”]

Feature selection in machine learning is selection of a feature which potentially would show some sort of pattern in the prediction analysis. Enron dataset consists two set of features - financial(salary, bonus, stock etc) and communication(to and from emails). Here I am more interested in emails those involve POIs. The assumption is that "Communication between POI and POI would be more frequent than between POI and non-POIs". So I want build a algorithm which would flag 'POI's among the population with good precision and recall score. For that, first I attempted to create two features, fractions of emails this person gets from POI(fraction_from_poi) and fraction of emails this person sends to POI(fraction_to_poi).

$\text{fraction_from_poi} = \text{number of emails this person gets from POI} / \text{total number of from messages}$

$\text{fraction_to_poi} = \text{number of emails this person sends to POI} / \text{total number sent messages}$

If email data is 'NaN', ratio is set to 0.


```

In [77]: #3 create new features
#new features: fraction_to_poi = fraction of emails sent to POIs, frac
tion_from_poi = fraction of emails received from POI
def dict_to_list(key,normalizer):
    new_list=[]

    for i in data_dict:
        if data_dict[i][key]=="NaN" or data_dict[i][normalizer]=="NaN"
:
            new_list.append(0.)
        elif data_dict[i][key]>=0:
            new_list.append(float(data_dict[i][key])/float(data_dict[i
][normalizer]))
    return new_list

### create two lists of new features
fraction_from_poi = dict_to_list("from_poi_to_this_person","to_message
s")
fraction_to_poi = dict_to_list("from_this_person_to_poi","from_message
s")

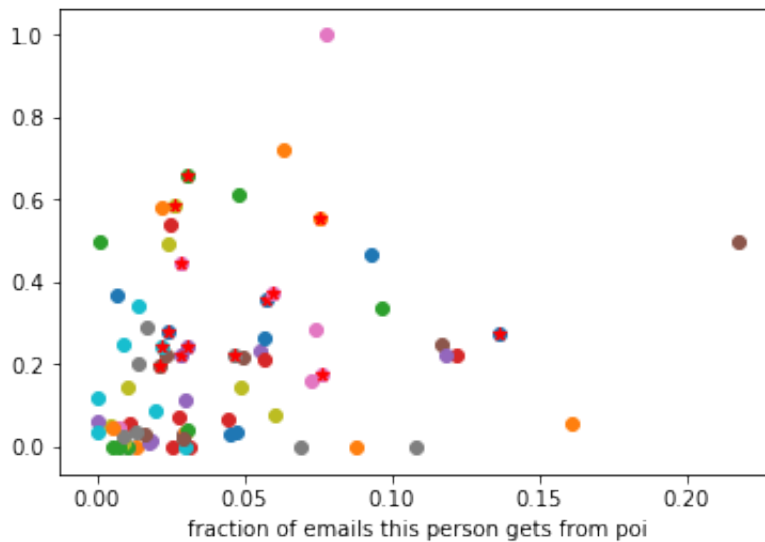
### insert new features into data_dict
count=0
for i in data_dict:
    data_dict[i]["fraction_from_poi"]=fraction_from_poi[count]
    data_dict[i]["fraction_to_poi"]=fraction_to_poi[count]
    count +=1

features_list = ["poi", "fraction_from_poi", "fraction_to_poi"]
    ### store to my_dataset for easy export below
my_dataset = data_dict

### these two lines extract the features specified in features_list
### and extract them from data_dict, returning a numpy array
data = featureFormat(my_dataset, features_list)

### plot new features
for point in data:
    from_poi = point[1]
    to_poi = point[2]
    plt.scatter( from_poi, to_poi )
    if point[0] == 1:
        plt.scatter(from_poi, to_poi, color="r", marker="*")
plt.xlabel("fraction of emails this person gets from poi")
plt.show()

```



```
In [78]: print features_list
['poi', 'fraction_from_poi', 'fraction_to_poi']
```

Feature Selection

This is the process where, we select and keep only impactful features in our dataset which contribute maximum information to the prediction variable. We do not want features which do not provide decent information. Having irrelevant features in our data decreases accuracy and quality of analysis. Univariate feature selection works by selecting the best features based on univariate statistical tests. SelectKBest removes all but the k highest scoring features. Here I used SelectKBest to see high scoring features.

Secondly, I used feature importance attribute of decision tree and this helped me to select features which would give me maximum precision and recall scores. I also used manual feature selection process where I tried adding each features one by one and calculated precision and recall scores. More importantly it was very interesting to see that features including 'shared_receipt_with_poi', 'exercised_stock_options' significantly increased precision and recall scores irrespective of the fact that they are not the top features in the selectKBest scores. Hence, features selected from this step were ["fraction_from_poi", "fraction_to_poi", "shared_receipt_with_poi", "exercised_stock_options"].

```

In [104]: from sklearn.feature_selection import SelectKBest, f_classif
features_list = ["poi", "salary", "bonus", "fraction_from_poi", "fraction_to_poi",
                 'deferral_payments', 'total_payments', 'loan_advances',
                 'restricted_stock_deferred',
                 'deferred_income', 'total_stock_value', 'expenses', 'exercised_stock_options',
                 'long_term_incentive', 'shared_receipt_with_poi', 'restricted_stock', 'director_fees']
data = featureFormat(my_dataset, features_list)

### split into labels and features (this line assumes that the first
### feature in the array is the label, which is why "poi" must always
### be first in features_list
labels, features = targetFeatureSplit(data)

### split data into training and testing datasets
#deploying feature selection
## used selectkbest and feature importance attribute of decision tree

selector = SelectKBest(f_classif, k=5)
selector.fit(features, labels)

# Get the raw p-values for each feature, and transform from p-values into scores
scores = -np.log10(selector.pvalues_)
indices = np.argsort(scores)[::-1]
print 'selectKBest scores ranking: '
for i in range(16):
    print "{} feature {} ({}).format(i+1,features_list[i+1],scores[indices[i]])

from sklearn import cross_validation
features_train, features_test, labels_train, labels_test = cross_validation.train_test_split(features, labels, test_size=0.1, random_state=42)

from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier()
clf.fit(features_train,labels_train)

importances = clf.feature_importances_
import numpy as np
indices = np.argsort(importances)[::-1]
print 'Feature importance Ranking: '
for i in range(14):
    print "{} feature {} ({}).format(i+1,features_list[i+1],importances[indices[i]])

```

```
selectKBest scores ranking:
1 feature salary (5.79736353414)
2 feature bonus (5.67658150738)
3 feature fraction_from_poi (5.0131174971)
4 feature fraction_to_poi (4.51801361022)
5 feature deferral_payments (4.12527738523)
6 feature total_payments (3.06560097119)
7 feature loan_advances (2.73390121244)
8 feature restricted_stock_deferred (2.57349505965)
9 feature deferred_income (2.46649213468)
10 feature total_stock_value (2.43956703429)
11 feature expenses (2.09833377594)
12 feature exercised_stock_options (1.86413139398)
13 feature long_term_incentive (1.12329211867)
14 feature shared_receipt_with_poi (0.827486110408)
15 feature restricted_stock (0.192462337735)
16 feature director_fees (0.0973697633378)
Feature importance Ranking:
1 feature salary (0.158290984378)
2 feature bonus (0.14622972935)
3 feature fraction_from_poi (0.14139568871)
4 feature fraction_to_poi (0.120901730257)
5 feature deferral_payments (0.118337314859)
6 feature total_payments (0.0955181169023)
7 feature loan_advances (0.0911276777792)
8 feature restricted_stock_deferred (0.0747826086957)
9 feature deferred_income (0.0534161490683)
10 feature total_stock_value (0.0)
11 feature expenses (0.0)
12 feature exercised_stock_options (0.0)
13 feature long_term_incentive (0.0)
14 feature shared_receipt_with_poi (0.0)
```

Algorithm selection and tuning

```

In [101]: features_list = ["poi", "exercised_stock_options", "fraction_from_poi",
    "fraction_to_poi", "shared_receipt_with_poi"]

##try Naive Bayes

t0 = time()
clf = GaussianNB()
clf.fit(features_train, labels_train)
pred = clf.predict(features_test)
print "NAIVE BAYES:"
print "Naive Bayes recall score:", (recall_score(labels_test,pred))
print "Naive Bayes precision score:", (precision_score(labels_test,pred))
#print accuracy #(clf.score(features_test, labels_test))
accuracy = accuracy_score(pred,labels_test)
print 'accuracy:', accuracy
print "NB algorithm time:", round(time()-t0, 3), 's'

###Adaboost
from sklearn.ensemble import AdaBoostClassifier
clf = AdaBoostClassifier(DecisionTreeClassifier(min_samples_split = 40
),
                        algorithm="SAMME",
                        n_estimators=200)
clf.fit(features_train,labels_train)
pred = clf.predict(features_test)

acc = accuracy_score(pred,labels_test)
print "ADABOOST:"
print acc
print "AB Recall Score:", str(recall_score(labels_test, pred))
print "AB Precision Score:", str(precision_score(labels_test, pred))

##try Decision tree
from sklearn.tree import DecisionTreeClassifier

t0 = time()

clf = DecisionTreeClassifier()
clf.fit(features_train,labels_train)
score = clf.score(features_test,labels_test)
print 'DECISION TREE:'
print "Decision tree algorithm time:", round(time()-t0, 3), "s"
print 'accuracy:',(score)
print 'recall score:', (recall_score(labels_test,pred))
print 'precision score:', (precision_score(labels_test,pred))

```

```
NAIVE BAYES:
Naive Bayes recall score: 0.5
Naive Bayes precision score: 0.181818181818
accuracy: 0.266666666667
NB algorithm time: 0.01 s
ADABOOST:
0.733333333333
AB Recall Score: 0.25
AB Precision Score: 0.5
DECISION TREE:
Decision tree algorithm time: 0.003 s
accuracy: 0.733333333333
recall score: 0.25
precision score: 0.5
```

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

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?

I chose decision tree as a final algorithm. Algorithm performance can be measured in a variety of ways, e.g. accuracy, precision, or recall. Since there were few POI subjects in this dataset, precision and recall score were considered better evaluator than accuracy. Naive Bayes, decision tree and adaboost algorithms were tried using four best features. Adaboost and Naive Bayes showed recall score of less than 0.3. Decision tree is more efficient in finding irregular decision boundaries and does not need feature scaling as splitting of the data is based on a threshold value.

Parameter tuning means the adjustment of parameters in a certain way that it helps in better performance of algorithm. If parameter is not tuned algorithm may not perform well. In general algorithms are not specifically tuned to any dataset.

In parameter tuning process, we need to evaluate and validate our data for each different combination of selected parameters and see how well the algorithm performs until we are satisfied with. If algorithm is not tuned well, it won't be trained well and it would not be able to make predictions on the unseen data. Here, to optimize its performance, parameters `min_sample_split` was varied manually and performance evaluated. We found algorithm performed well when `min_sample_split` was kept 3.

```

In [102]: ### use manual tuning parameter min_samples_split
from sklearn.tree import DecisionTreeClassifier
t0 = time()
clf = DecisionTreeClassifier(min_samples_split = 3)
clf = clf.fit(features_train,labels_train)
pred= clf.predict(features_test)
print("done in %0.3fs" % (time() - t0))

acc=accuracy_score(labels_test, pred)

print "Validating algorithm:"
print "accuracy after tuning = ", acc

# function for calculation ratio of true positives
# out of all positives (true + false)
print 'precision = ', precision_score(labels_test,pred)

# function for calculation ratio of true positives
# out of true positives and false negatives
print 'recall = ', recall_score(labels_test,pred)

### dump your classifier, dataset and features_list so
### anyone can run/check your results
pickle.dump(clf, open("my_classifier.pkl", "w") )
pickle.dump(data_dict, open("my_dataset.pkl", "w") )
pickle.dump(features_list, open("my_feature_list.pkl", "w") )

```

done in 0.003s

Validating algorithm:

accuracy after tuning = 0.733333333333

precision = 0.5

recall = 0.5

Analysis Validation and Performance

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 where we determine the robustness of our predictive models, i.e., to test how well our model is trained.

A classic mistake in validation process is called over-fitting, where the model is trained and performs too good on the training dataset, but is actually worse on the cross-validation and test datasets. Besides, also if algorithm is tested in the same dataset as it was trained on it would give very high performance.

I used k-fold cross-validation. K-fold cross-validation method randomly divides the dataset into a test and training set k different times and let you decide how many trials to average at the end. You can also independently choose how large each test set is. In each trial, one of the k subsets is used as the test set and the other k-1 subsets are used as a training set.

```
In [103]: ### features_list is a list of strings, each of which is a feature name
### first feature must be "poi", as this will be singled out as the label
features_list = ["poi", "exercised_stock_options", "fraction_from_poi",
, "fraction_to_poi", "shared_receipt_with_poi"]

### store to my_dataset for easy export below
my_dataset = data_dict

### these two lines extract the features specified in features_list
### and extract them from data_dict, returning a numpy array
data = featureFormat(my_dataset, features_list)

### split into labels and features (this line assumes that the first
### feature in the array is the label, which is why "poi" must always
### be first in features_list
labels, features = targetFeatureSplit(data)

### machine learning goes here!
### please name your classifier clf for easy export below

### deploying feature selection
from sklearn import cross_validation
features_train, features_test, labels_train, labels_test = cross_validation.train_test_split(features, labels, test_size=0.1, random_state=42)
```



```

### use KFold for split and validate algorithm
from sklearn.cross_validation import KFold
kf=KFold(len(labels),3)
for train_indices, test_indices in kf:
    #make training and testing sets
    features_train= [features[ii] for ii in train_indices]
    features_test= [features[ii] for ii in test_indices]
    labels_train=[labels[ii] for ii in train_indices]
    labels_test=[labels[ii] for ii in test_indices]

from sklearn.tree import DecisionTreeClassifier

t0 = time()

clf = DecisionTreeClassifier()
clf.fit(features_train,labels_train)
score = clf.score(features_test,labels_test)
print 'accuracy before tuning ', score

print "Decision tree algorithm time:", round(time()-t0, 3), "s"

### use manual tuning parameter min_samples_split
t0 = time()
clf = DecisionTreeClassifier(min_samples_split = 3)
clf = clf.fit(features_train,labels_train)
pred= clf.predict(features_test)
print("done in %0.3fs" % (time() - t0))

acc=accuracy_score(labels_test, pred)

print "Validating algorithm:"
print "accuracy after tuning = ", acc

# function for calculation ratio of true positives
# out of all positives (true + false)
print 'precision = ', precision_score(labels_test,pred)

# function for calculation ratio of true positives
# out of true positives and false negatives
print 'recall = ', recall_score(labels_test,pred)

### dump your classifier, dataset and features_list so
### anyone can run/check your results
pickle.dump(clf, open("my_classifier.pkl", "w") )
pickle.dump(data_dict, open("my_dataset.pkl", "w") )
pickle.dump(features_list, open("my_feature_list.pkl", "w") )

```

```
accuracy before tuning 0.875
Decision tree algorithm time: 0.003 s
done in 0.003s
Validating algorithm:
accuracy after tuning = 0.85
precision = 0.4
recall = 0.4
```

```
In [106]: data = featureFormat(my_dataset, features_list, sort_keys = True)
          print "Number of observations: {}".format(len(features))

Number of observations: 144
```

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.

I chose Decision Tree as a final algorithm to build a model to predict POIs from the population. In this case we used precision and recall as a measure of model performance. A good precision means if a person is identified as POI in the test will infact be POI. Low precision score can give false positive results which means innocent people can be flagged as POI.

If the precision score = 0.4, it means 40 percent of POI identified in the test would be the real POI whereas 60% of the time this test could flag a non POI. A good recall means relevant subjects from the population are all identified. If the recall score = 0.4, it means 60 percent of POI will go unidentified. In our case of POI identifier, it is more important to not miss any POIs than to flag non-POIs. So we prefer high recall. Decision tree performed best with the recall and precision value of 0.4. However, On checking the performance of my algorithm in tester.py's cross validation I got precision and recall of 0.46 and 0.31 respectively.

References:

https://en.wikipedia.org/wiki/Enron_Corpus (https://en.wikipedia.org/wiki/Enron_Corpus)

<https://www.cs.cmu.edu/~enron/> (<https://www.cs.cmu.edu/~enron/>)

<https://www.technologyreview.com/s/515801/the-immortal-life-of-the-enron-e-mails/>
(<https://www.technologyreview.com/s/515801/the-immortal-life-of-the-enron-e-mails/>)

<https://classroom.udacity.com/nanodegrees/nd002/parts/0021345409/modules/317428862475461/lessons/3>
(<https://classroom.udacity.com/nanodegrees/nd002/parts/0021345409/modules/317428862475461/lessons/3>)

<https://discussions.udacity.com/c/nd002-intro-to-machine-learning>
(<https://discussions.udacity.com/c/nd002-intro-to-machine-learning>)

http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedShuffleSplit.html
(http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedShuffleSplit.html)

<http://rushdishams.blogspot.co.id/2011/03/precision-and-recall.html>
(<http://rushdishams.blogspot.co.id/2011/03/precision-and-recall.html>)

<https://www.cs.cmu.edu/~schneide/tut5/node42.html>
(<https://www.cs.cmu.edu/~schneide/tut5/node42.html>)

In []: