Machine Learning Goal of Project “POI Identifier” summary:

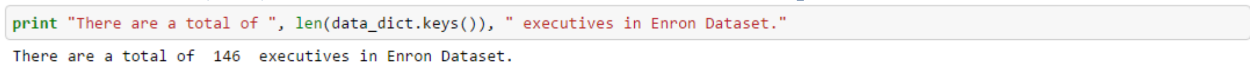
1. Summarize for us the goal of this project and how machine learning is useful in trying to accomplish it. As part of your answer, give some background on the dataset and how it can be used to answer the project question. Were there any outliers in the data when you got it, and how did you handle those? [relevant rubric items: “data exploration”, “outlier investigation”]

The Enron data was originally collected at Enron Corporation headquarters in Houston during two weeks in May 2002 by Joe Bartling, [3] a litigation support and data analysis contractor working for Aspen Systems, now Lockheed Martin, whom the Federal Energy Regulatory Commission (FERC) had hired to preserve and collect the vast amounts of data in the wake of the Enron Bankruptcy in December 2001.

The corpus is unique in that it is one of the only publicly available mass collections of real emails easily available for study.

The goal of this project is to build a person of interest identifier to identify persons-of-interest (POI’s) using a machine learning algorithm. POI’s were ‘individuals who were indicted, reached a settlement, or plea deal with the government, or testified in exchange for prosecution immunity.’ Financial compensation data and aggregate email statistics from the Enron Corpus were used as features for prediction.

The dataset contained 146 records with 14 financial features, 6 email features, and 1 labeled feature (POI). Of the 146 records, 18 were labeled, as persons of interest.



```
print "There are a total of ", len(data_dict.keys()), " executives in Enron Dataset."
```

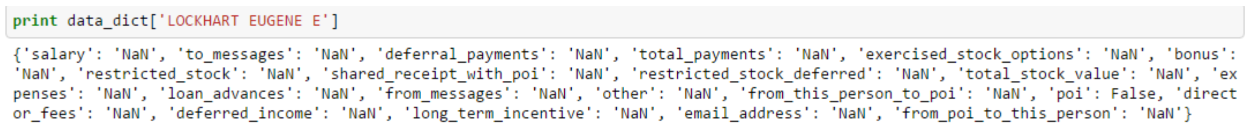
```
There are a total of  146  executives in Enron Dataset.
```

Through exploratory data analysis and also by reviewing the file, found that the following fields were outliers.

TOTAL: This was an outlier as it was totaling of the financial statistics from the financial data.

THE TRAVEL AGENCY IN THE PARK: This record did not represent an individual.

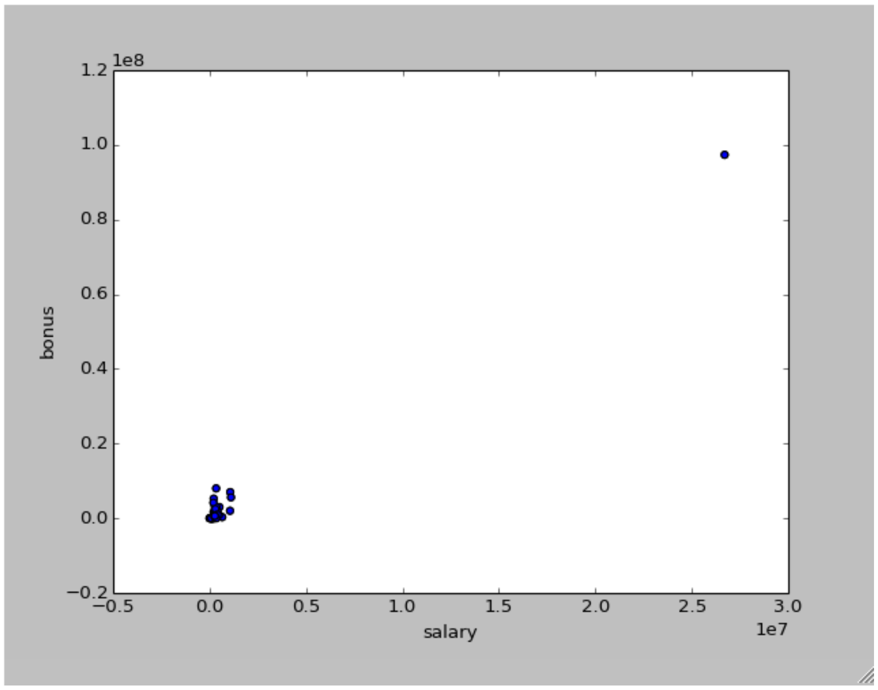
LOCKHART EUGENE E: was removed during data processing since this row had no entries for any feature and had lot of NaN values



```
print data_dict['LOCKHART EUGENE E']
```

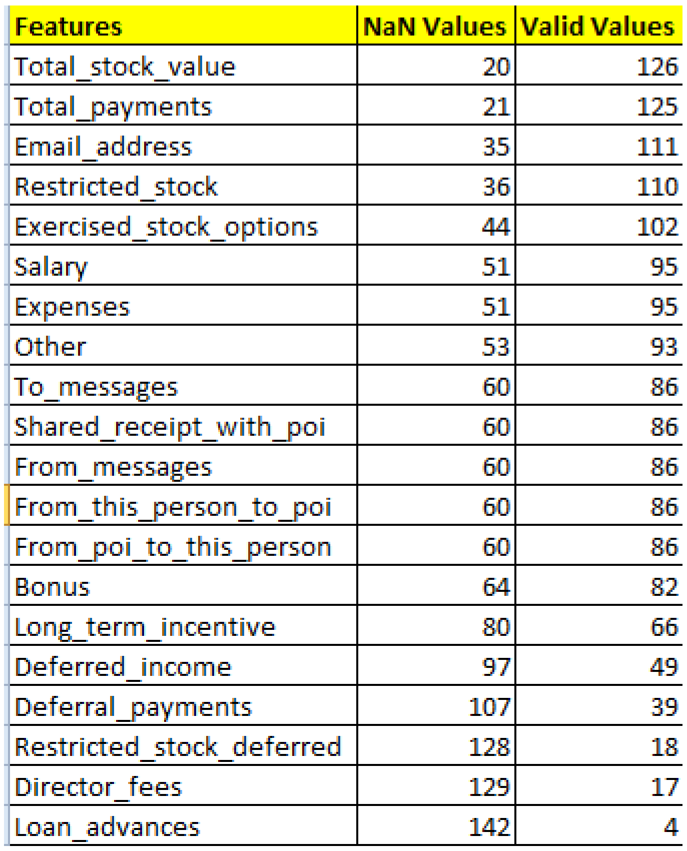
```
{'salary': 'NaN', 'to_messages': 'NaN', 'deferral_payments': 'NaN', 'total_payments': 'NaN', 'exercised_stock_options': 'NaN', 'bonus':
'NaN', 'restricted_stock': 'NaN', 'shared_receipt_with_poi': 'NaN', 'restricted_stock_deferred': 'NaN', 'total_stock_value': 'NaN', 'ex
penses': 'NaN', 'loan_advances': 'NaN', 'from_messages': 'NaN', 'other': 'NaN', 'from_this_person_to_poi': 'NaN', 'poi': False, 'direct
or_fees': 'NaN', 'deferred_income': 'NaN', 'long_term_incentive': 'NaN', 'email_address': 'NaN', 'from_poi_to_this_person': 'NaN'}
```

```
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]
    mpplt.scatter( salary, bonus )

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



After removing outliers there were a total of 143 records.

Also here is the list of features and their corresponding NaN count.

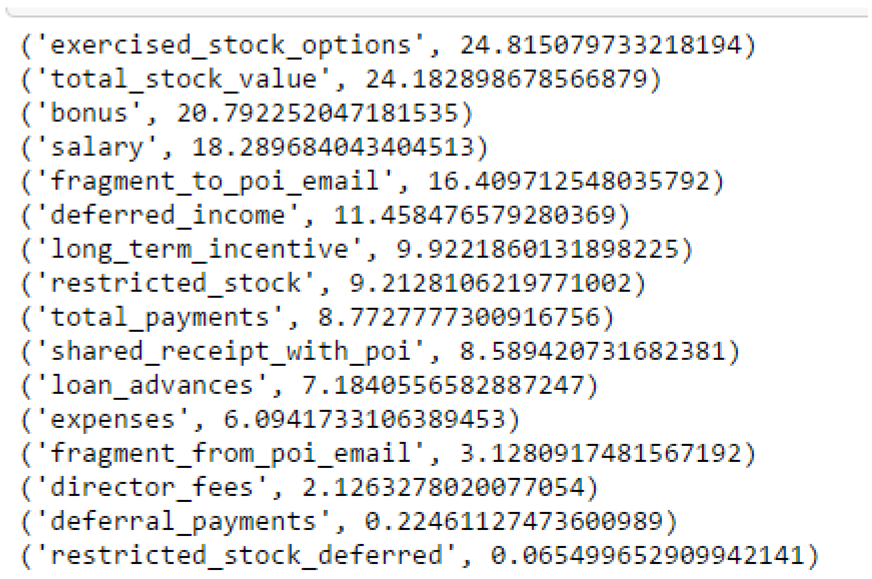


Features	NaN Values	Valid Values
Total_stock_value	20	126
Total_payments	21	125
Email_address	35	111
Restricted_stock	36	110
Exercised_stock_options	44	102
Salary	51	95
Expenses	51	95
Other	53	93
To_messages	60	86
Shared_receipt_with_poi	60	86
From_messages	60	86
From_this_person_to_poi	60	86
From_poi_to_this_person	60	86
Bonus	64	82
Long_term_incentive	80	66
Deferred_income	97	49
Deferral_payments	107	39
Restricted_stock_deferred	128	18
Director_fees	129	17
Loan_advances	142	4

Any features with at least more than half the values NaN are considered for their contribution towards identifying the POI.

2. What features did you end up using in your POI identifier, and what selection process did you use to pick them? Did you have to do any scaling? Why or why not? As part of the assignment, you should attempt to engineer your own feature that does not come ready-made in the dataset -- explain what feature you tried to make, and the rationale behind it. (You do not necessarily have to use it in the final analysis, only engineer and test it.) In your feature selection step, if you used an algorithm like a decision tree, please also give the feature importance of the features that you use, and if you used an automated feature selection function like SelectKBest, please report the feature scores and reasons for your choice of parameter values. [relevant rubric items: “create new features”, “properly scale features”, “intelligently select feature”]

To select the most relevant features, I used scikit-learn's automated univariate feature selection algorithm SelectKBest to obtain associated score (relevancy) of each feature for features in the features list. The features and their associated scores are listed below

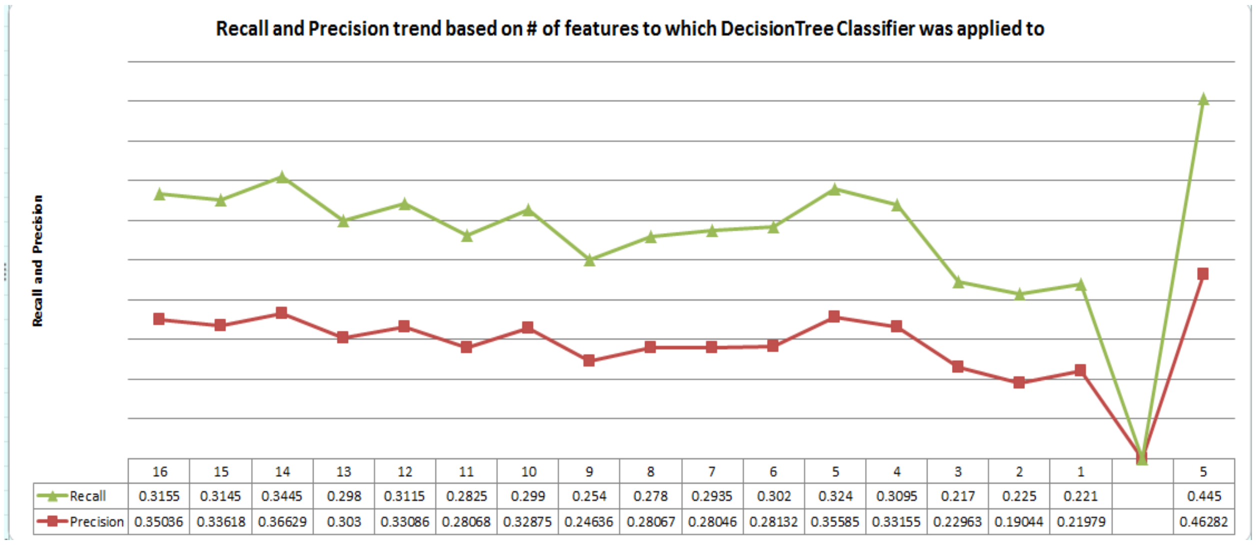


```
('exercised_stock_options', 24.815079733218194)
('total_stock_value', 24.182898678566879)
('bonus', 20.792252047181535)
('salary', 18.289684043404513)
('fragment_to_poi_email', 16.409712548035792)
('deferred_income', 11.458476579280369)
('long_term_incentive', 9.9221860131898225)
('restricted_stock', 9.2128106219771002)
('total_payments', 8.7727777300916756)
('shared_receipt_with_poi', 8.589420731682381)
('loan_advances', 7.1840556582887247)
('expenses', 6.0941733106389453)
('fragment_from_poi_email', 3.1280917481567192)
('director_fees', 2.1263278020077054)
('deferral_payments', 0.22461127473600989)
('restricted_stock_deferred', 0.065499652909942141)
```

I picked total_stock_value, bonus, salary and exercised_stock_options, one of the two features that I created which is fragment_to_poi_email (which is fraction of email that was sent by the person to POI) and the shared_receipt_with_poi.

Applied the Decision Tree on the StratifiedShuffleSplit beginning with all the features and then eliminated one by one calculating the precision and recall for each set of features and then identified those features elimination of which caused a dip in the recall value and added them back and came up with the final five.

Below is the trend of Recall and Precision based on the different features that were used in the DecisionTree classifier.



	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1		5
Recall	0.3155	0.3145	0.3445	0.298	0.3115	0.2825	0.299	0.254	0.278	0.2935	0.302	0.324	0.3095	0.217	0.225	0.221		0.445
Precision	0.35036	0.33618	0.36629	0.303	0.33086	0.28068	0.32875	0.24636	0.28067	0.28046	0.28132	0.35585	0.33155	0.22963	0.19044	0.21979		0.46282

A person with higher salary or bonus alone does not make them a POI, but financial gain along with intent to commit fraud does make them a POI which led to the below two feature picks.

The theory behind the creation of the two new features fragment_to_poi_email and fragment_from_poi_email is that there could be a link between the total # of contacts a person had with POI and that the person themselves being POI.

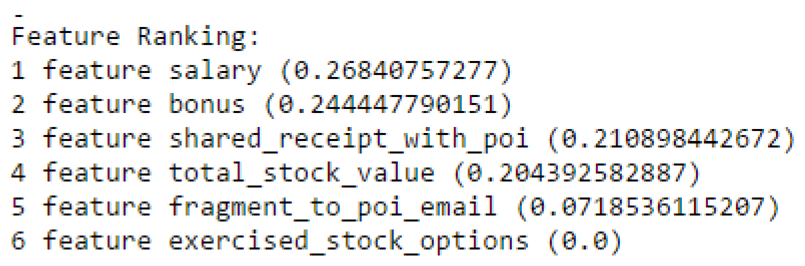
Prior to training the machine learning algorithm classifiers, I scaled all features using a MinMaxScaler for local testing. As the features list comprised of email messages and financial data, which varied by units and also two totally different entities. Feature-scaling ensured the features would be weighted evenly for the applicable classifiers.

The recall and precision with and without the two feature fragment_to_poi_email and fragment_from_poi_email using DecisionTree classifier are as below and shows how the features contribute towards a higher recall.

When we remove the fragment_to_poi_email from the list of features to which the classifier is applied the Recall goes down to 0.217 from 0.3095. When we remove fragment_to_poi_email from the list of features the Recall value goes up from 0.217 to 0.225.

I have attached (RecallAndPrecision.xlsx) the feature importance of DecisionTree classifier application process for each set of features.

The feature importance of the final set of feature list is as below



```
-
Feature Ranking:
1 feature salary (0.26840757277)
2 feature bonus (0.244447790151)
3 feature shared_receipt_with_poi (0.210898442672)
4 feature total_stock_value (0.204392582887)
5 feature fragment_to_poi_email (0.0718536115207)
6 feature exercised_stock_options (0.0)
```

What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms? [relevant rubric item: “pick an algorithm”]

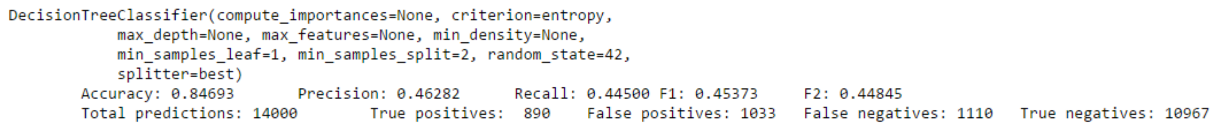
I ended up picking DecisionTree. The other Algorithms that I tried were KNeighbors, Naïve Bayes, and GridSearchCV with KNeighbors.

KNeighbors had the highest precision but slightly lower Recall compared to DecisionTree. Also during the iterations with the Cross Validators DecisionTree produced a better result compared to KNeighbors in that the Recall and Precision were more consistent with DecisionTree compared to KNeighbors. Naïve Bayes also performed well but since there were not any parameters to tune, I did not pick that one for final two algorithm compare.

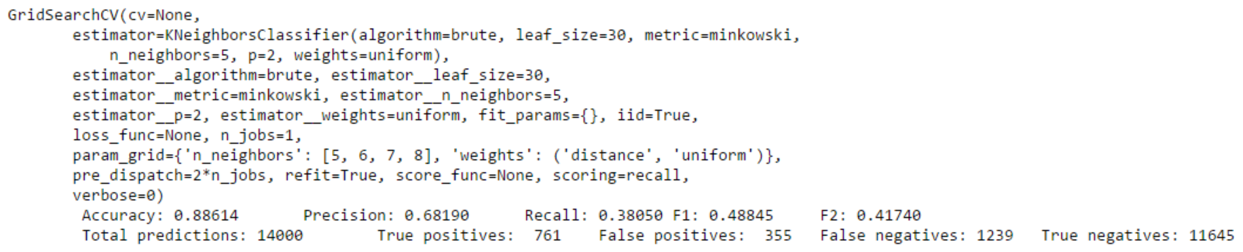
Since the goal is to identify POI's, we want to identify as many relevant POI's as we can (that's recall) and avoid having to sort through irrelevant data (that's precision).

In a pattern recognition and information retrieval with binary classification, the goal is typically to identify a small number of matches (POI's) in a large dataset. Because of this asymmetry, it is in fact much more difficult to get a good precision (how many of the positively classified were relevant) than a good specificity (how good a test is at avoiding false alarms) while keeping the sensitivity/recall constant (how good a test is at detecting the positives). Since most of the data/documents are irrelevant, we will have many more occasions for false alarms than true positives and these false alarms can swamp the correct results even if the classifier has impressive accuracy on a balanced test set. The object is to identify POI's and not exclude innocent people.

Based on the above reasoning, picked DecisionTree which has lower precision but recall is higher than KNeighbors. Both precision and recall are > 0.3 for DecisionTree.



```
DecisionTreeClassifier(compute_importances=None, criterion=entropy,
            max_depth=None, max_features=None, min_density=None,
            min_samples_leaf=1, min_samples_split=2, random_state=42,
            splitter=best)
        Accuracy: 0.84693       Precision: 0.46282      Recall: 0.44500 F1: 0.45373     F2: 0.44845
        Total predictions: 14000        True positives:  890    False positives: 1033   False negatives: 1110   True negatives: 10967
```



```
GridSearchCV(cv=None,
       estimator=KNeighborsClassifier(algorithm=brute, leaf_size=30, metric=minkowski,
           n_neighbors=5, p=2, weights=uniform),
       estimator__algorithm=brute, estimator__leaf_size=30,
       estimator__metric=minkowski, estimator__n_neighbors=5,
       estimator__p=2, estimator__weights=uniform, fit_params={}, iid=True,
       loss_func=None, n_jobs=1,
       param_grid={'n_neighbors': [5, 6, 7, 8], 'weights': ('distance', 'uniform')},
       pre_dispatch=2*n_jobs, refit=True, score_func=None, scoring=recall,
       verbose=0)
        Accuracy: 0.88614       Precision: 0.68190      Recall: 0.38050 F1: 0.48845     F2: 0.41740
        Total predictions: 14000        True positives:  761    False positives:  355   False negatives: 1239   True negatives: 11645
```

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? (Some algorithms do not have parameters that you need to tune -- if this is the case for the one you picked, identify and briefly explain how you would have done it for the model that was not your final choice or a different model that does utilize parameter tuning, e.g. a decision tree classifier). [relevant rubric item: "tune the algorithm"]

   The algorithms in scikit learn come with a number of parameters that have an impact on how the algorithm is being processed. Tuning the parameters is the process of setting the parameters to obtain the best possible result out of the classifiers. In this case tuning the algorithm is to find the balance between precision and recall.

   I tuned KNeighbors using (parameters = {'n_neighbors':[5,6,7,8] }) by using trial and error method.

```
GridSearchCV(cv=None,
       estimator=KNeighborsClassifier(algorithm=brute, leaf_size=30, metric=minkowski,
           n_neighbors=5, p=2, weights=uniform),
       estimator__algorithm=brute, estimator__leaf_size=30,
       estimator__metric=minkowski, estimator__n_neighbors=5,
       estimator__p=2, estimator__weights=uniform, fit_params={}, iid=True,
       loss_func=None, n_jobs=1,
       param_grid={'n_neighbors': [5, 6, 7, 8], 'weights': ('distance', 'uniform')},
       pre_dispatch=2*n_jobs, refit=True, score_func=None, scoring=recall,
       verbose=0)
        Accuracy: 0.88614       Precision: 0.68190      Recall: 0.38050 F1: 0.48845     F2: 0.41740
        Total predictions: 14000        True positives:  761    False positives:  355   False negatives: 1239   True negatives: 11645
```

I tuned the DecisionTree by setting the Criterion to “Entropy” and random_state to 42 after trying out diff parameters which improved both precision and recall value.

```
DecisionTreeClassifier(compute_importances=None, criterion=entropy,
            max_depth=None, max_features=None, min_density=None,
            min_samples_leaf=1, min_samples_split=2, random_state=42,
            splitter=best)
        Accuracy: 0.84693       Precision: 0.46282      Recall: 0.44500 F1: 0.45373     F2: 0.44845
        Total predictions: 14000        True positives:  890    False positives: 1033   False negatives: 1110   True negatives: 10967
```

Other tuning options I considered for the DecisionTree classifier are as below

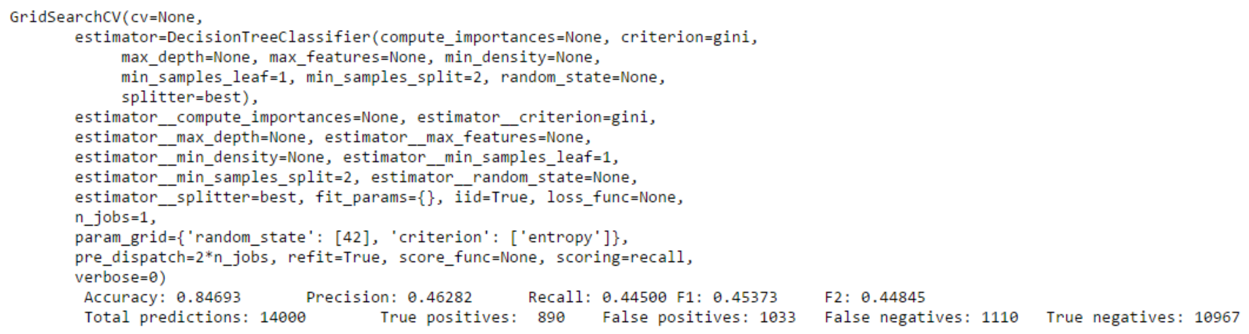
(1)

```
#for testing different classifier options
clf = DecisionTreeClassifier(min_samples_split = 2, max_depth = 8,splitter = "best", random_state=42)
parameters = {'criterion':['entropy'],'random_state':[42]}
#clf = DecisionTreeClassifier(criterion = "entropy", random_state=42)

clf = GridSearchCV(clf, parameters,scoring="recall")
```

```
GridSearchCV(cv=None,
       estimator=DecisionTreeClassifier(compute_importances=None, criterion=gini, max_depth=8,
            max_features=None, min_density=None, min_samples_leaf=1,
            min_samples_split=2, random_state=42, splitter=best),
       estimator__compute_importances=None, estimator__criterion=gini,
       estimator__max_depth=8, estimator__max_features=None,
       estimator__min_density=None, estimator__min_samples_leaf=1,
       estimator__min_samples_split=2, estimator__random_state=42,
       estimator__splitter=best, fit_params={}, iid=True, loss_func=None,
       n_jobs=1,
       param_grid={'random_state': [42], 'criterion': ['entropy']},
       pre_dispatch=2*n_jobs, refit=True, score_func=None, scoring=recall,
       verbose=0)
        Accuracy: 0.84679       Precision: 0.46230      Recall: 0.44450 F1: 0.45322     F2: 0.44795
        Total predictions: 14000        True positives:  889    False positives: 1034   False negatives: 1111   True negatives: 10966
```

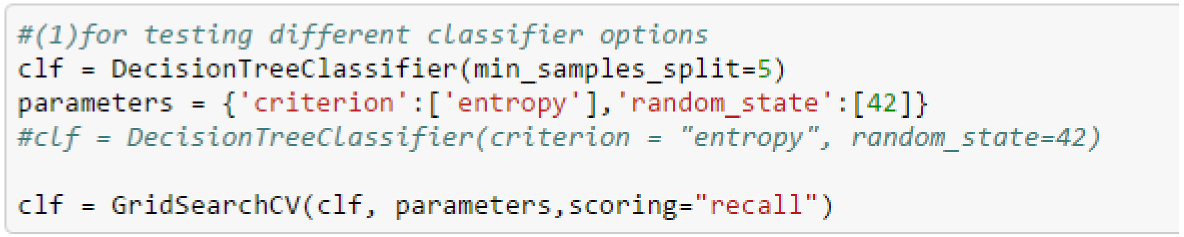
(2)

```
#(1)for testing different classifier options
clf = DecisionTreeClassifier(splitter = "best")
parameters = {'criterion':['entropy'],'random_state':[42]}
#clf = DecisionTreeClassifier(criterion = "entropy", random_state=42)

clf = GridSearchCV(clf, parameters,scoring="recall")
```

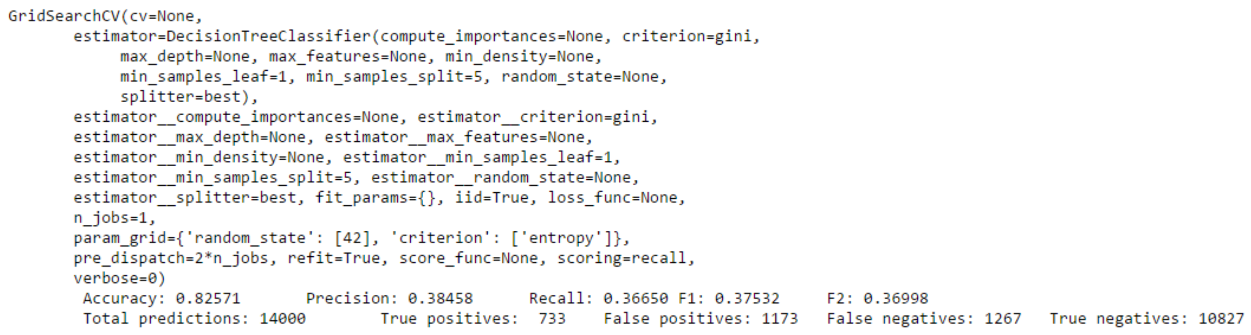


```
GridSearchCV(cv=None,
       estimator=DecisionTreeClassifier(compute_importances=None, criterion=gini,
            max_depth=None, max_features=None, min_density=None,
            min_samples_leaf=1, min_samples_split=2, random_state=None,
            splitter=best),
       estimator__compute_importances=None, estimator__criterion=gini,
       estimator__max_depth=None, estimator__max_features=None,
       estimator__min_density=None, estimator__min_samples_leaf=1,
       estimator__min_samples_split=2, estimator__random_state=None,
       estimator__splitter=best, fit_params={}, iid=True, loss_func=None,
       n_jobs=1,
       param_grid={'random_state': [42], 'criterion': ['entropy']},
       pre_dispatch=2*n_jobs, refit=True, score_func=None, scoring=recall,
       verbose=0)
        Accuracy: 0.84693       Precision: 0.46282      Recall: 0.44500 F1: 0.45373     F2: 0.44845
        Total predictions: 14000        True positives:  890    False positives: 1033   False negatives: 1110   True negatives: 10967
```

(3)

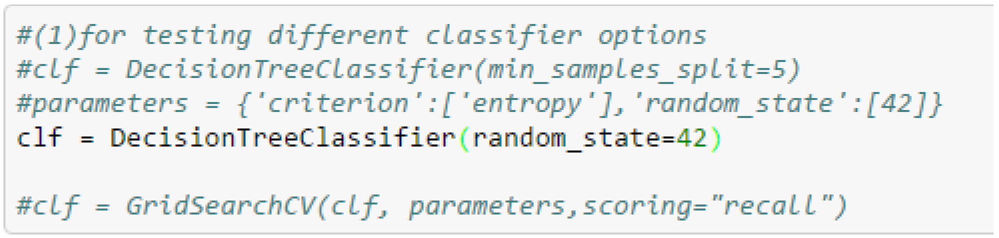


```
#(1)for testing different classifier options
clf = DecisionTreeClassifier(min_samples_split=5)
parameters = {'criterion':['entropy'],'random_state':[42]}
#clf = DecisionTreeClassifier(criterion = "entropy", random_state=42)

clf = GridSearchCV(clf, parameters,scoring="recall")
```

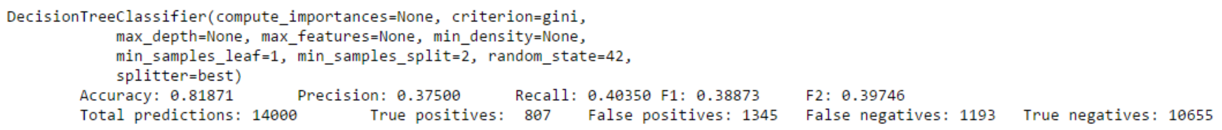


```
GridSearchCV(cv=None,
       estimator=DecisionTreeClassifier(compute_importances=None, criterion=gini,
            max_depth=None, max_features=None, min_density=None,
            min_samples_leaf=1, min_samples_split=5, random_state=None,
            splitter=best),
       estimator__compute_importances=None, estimator__criterion=gini,
       estimator__max_depth=None, estimator__max_features=None,
       estimator__min_density=None, estimator__min_samples_leaf=1,
       estimator__min_samples_split=5, estimator__random_state=None,
       estimator__splitter=best, fit_params={}, iid=True, loss_func=None,
       n_jobs=1,
       param_grid={'random_state': [42], 'criterion': ['entropy']},
       pre_dispatch=2*n_jobs, refit=True, score_func=None, scoring=recall,
       verbose=0)
        Accuracy: 0.82571       Precision: 0.38458      Recall: 0.36650 F1: 0.37532     F2: 0.36998
        Total predictions: 14000        True positives:  733    False positives: 1173   False negatives: 1267   True negatives: 10827
```

(4)

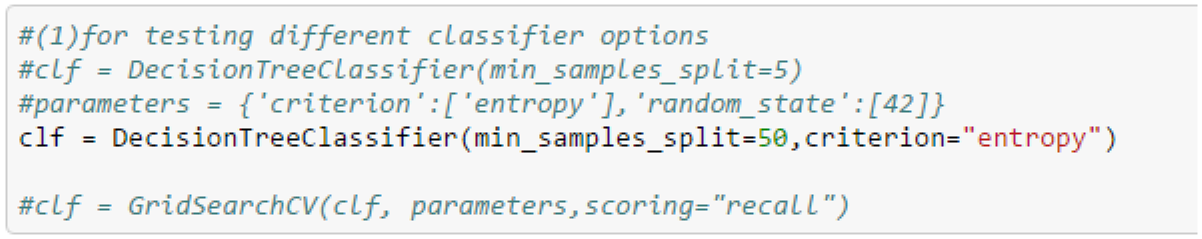


```
#(1)for testing different classifier options
#clf = DecisionTreeClassifier(min_samples_split=5)
#parameters = {'criterion':['entropy'],'random_state':[42]}
clf = DecisionTreeClassifier(random_state=42)

#clf = GridSearchCV(clf, parameters,scoring="recall")
```

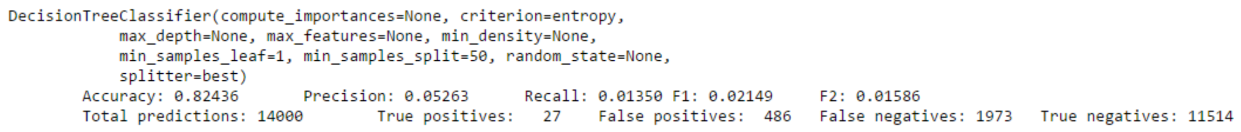


```
DecisionTreeClassifier(compute_importances=None, criterion=gini,
            max_depth=None, max_features=None, min_density=None,
            min_samples_leaf=1, min_samples_split=2, random_state=42,
            splitter=best)
        Accuracy: 0.81871       Precision: 0.37500      Recall: 0.40350 F1: 0.38873     F2: 0.39746
        Total predictions: 14000        True positives:  807    False positives: 1345   False negatives: 1193   True negatives: 10655
```

(5)

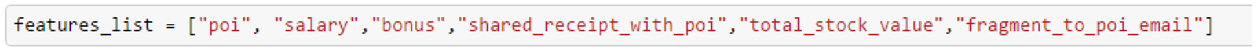


```
#(1)for testing different classifier options
#clf = DecisionTreeClassifier(min_samples_split=5)
#parameters = {'criterion':['entropy'],'random_state':[42]}
clf = DecisionTreeClassifier(min_samples_split=50,criterion="entropy")

#clf = GridSearchCV(clf, parameters,scoring="recall")
```

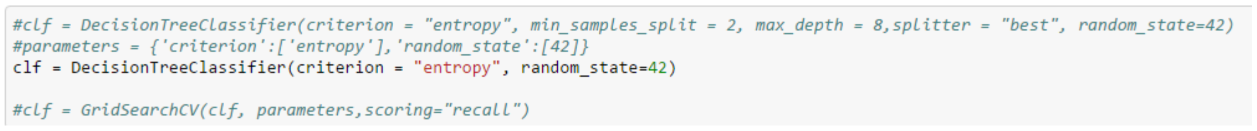


```
DecisionTreeClassifier(compute_importances=None, criterion=entropy,
            max_depth=None, max_features=None, min_density=None,
            min_samples_leaf=1, min_samples_split=50, random_state=None,
            splitter=best)
        Accuracy: 0.82436       Precision: 0.05263      Recall: 0.01350 F1: 0.02149     F2: 0.01586
        Total predictions: 14000        True positives:   27    False positives:  486   False negatives: 1973   True negatives: 11514
```

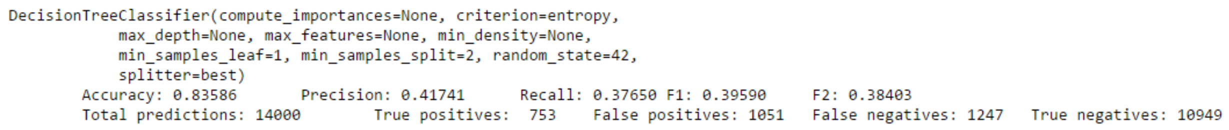
(6)



```
features_list = ["poi", "salary","bonus","shared_receipt_with_poi","total_stock_value","fragment_to_poi_email"]
```



```
#clf = DecisionTreeClassifier(criterion = "entropy", min_samples_split = 2, max_depth = 8,splitter = "best", random_state=42)
#parameters = {'criterion':['entropy'],'random_state':[42]}
clf = DecisionTreeClassifier(criterion = "entropy", random_state=42)

#clf = GridSearchCV(clf, parameters,scoring="recall")
```



```
DecisionTreeClassifier(compute_importances=None, criterion=entropy,
            max_depth=None, max_features=None, min_density=None,
            min_samples_leaf=1, min_samples_split=2, random_state=42,
            splitter=best)
        Accuracy: 0.83586       Precision: 0.41741      Recall: 0.37650 F1: 0.39590     F2: 0.38403
        Total predictions: 14000        True positives:  753    False positives: 1051   False negatives: 1247   True negatives: 10949
```

5. What is validation, and what's a classic mistake you can make if you do it wrong? How did you validate your analysis? [relevant rubric item: "validation strategy"]

Validation is the process of testing the classifier on a subset of training data that was kept separate from the data that was used to train the algorithm (testing data).

A classic mistake would be tuning the model to be able to predict the training data very well, but then the model performing poorly on unseen out-of-sample testing data. This is called overfitting. One of the major goals in validation is to avoid overfitting, which can be accomplished through a process called cross-validation.

Cross-validation is the process of randomly splitting the data into training and testing data. Then the model can train on the training data, and be validated on the testing data.

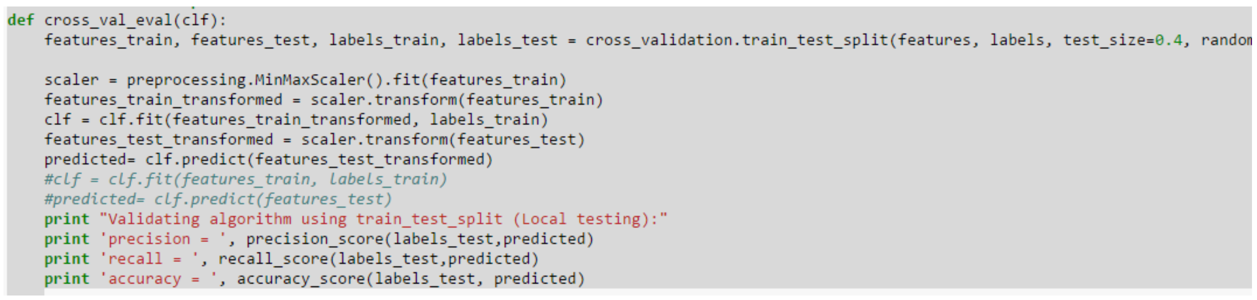
The StratifiedShuffleSplit model which is a mix of K Fold cross validation and ShuffleSplit splits the data into new training and testing splits using 1000 randomized stratified cross-validation splits and then trains and tests the algorithm on each of those splits

A similar validation procedure was used in tester.py to evaluate the resulting final models that were selected.

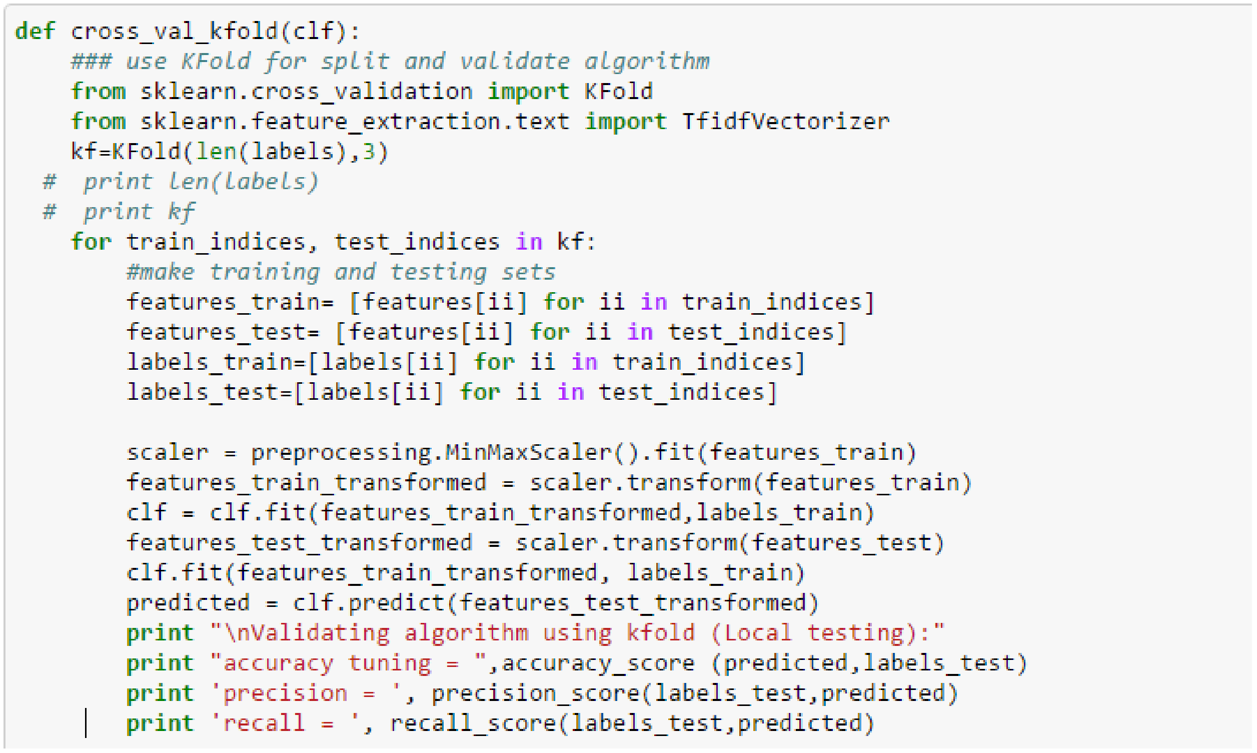
For local testing I scaled all features using a MinMaxScaler.

I used 3 cross validators to validate the algorithm, train_test_split, kFold and Stratified Shuffle split to split the features

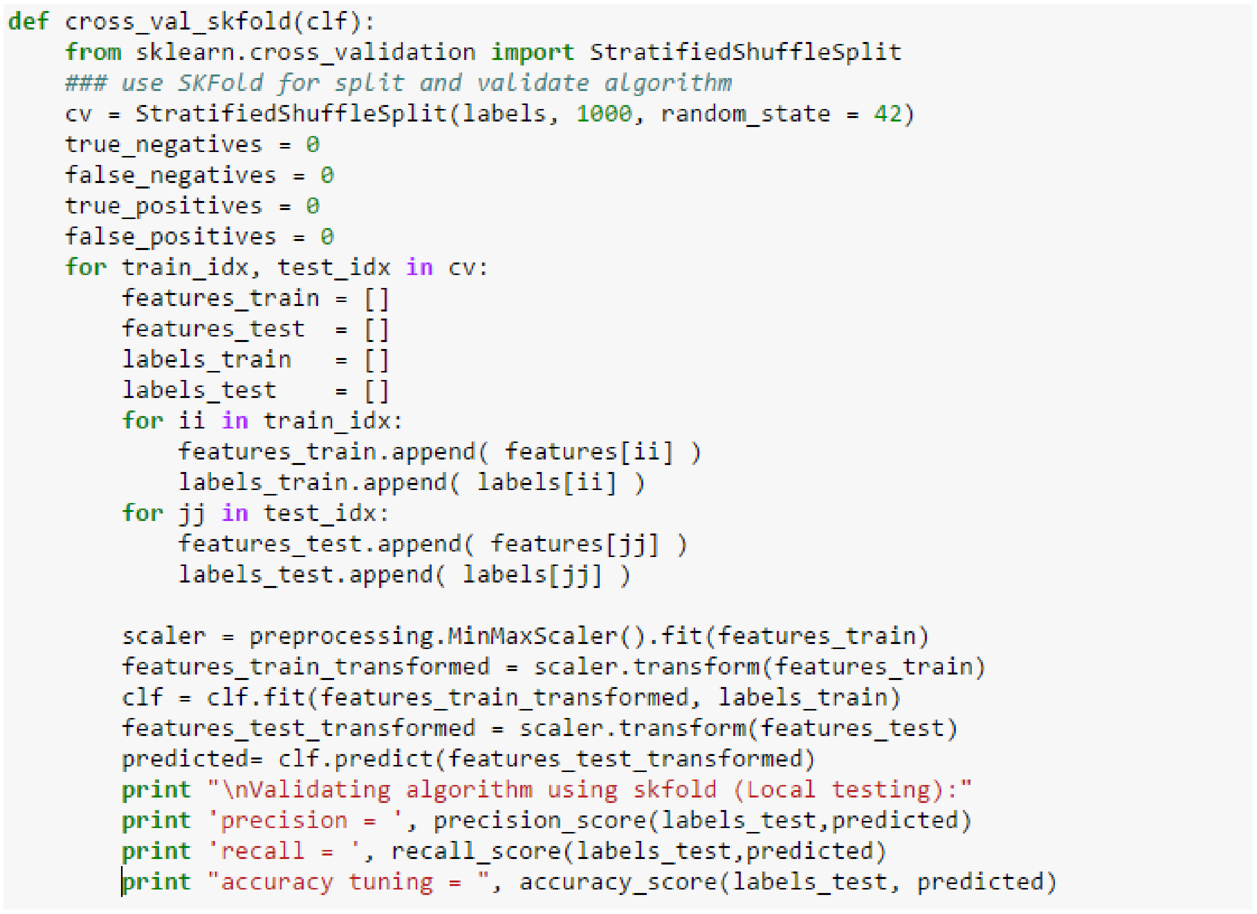
The StratifiedShuffleSplit is appropriate validation method for this dataset because the dataset is small and unbalanced. Reserving just 10% of the training data for testing would not provide enough data for robust training.



```
def cross_val_eval(clf):
    features_train, features_test, labels_train, labels_test = cross_validation.train_test_split(features, labels, test_size=0.4, random

    scaler = preprocessing.MinMaxScaler().fit(features_train)
    features_train_transformed = scaler.transform(features_train)
    clf = clf.fit(features_train_transformed, labels_train)
    features_test_transformed = scaler.transform(features_test)
    predicted= clf.predict(features_test_transformed)
    #clf = clf.fit(features_train, labels_train)
    #predicted= clf.predict(features_test)
    print "Validating algorithm using train_test_split (Local testing):"
    print 'precision = ', precision_score(labels_test,predicted)
    print 'recall = ', recall_score(labels_test,predicted)
    print 'accuracy = ', accuracy_score(labels_test, predicted)
```



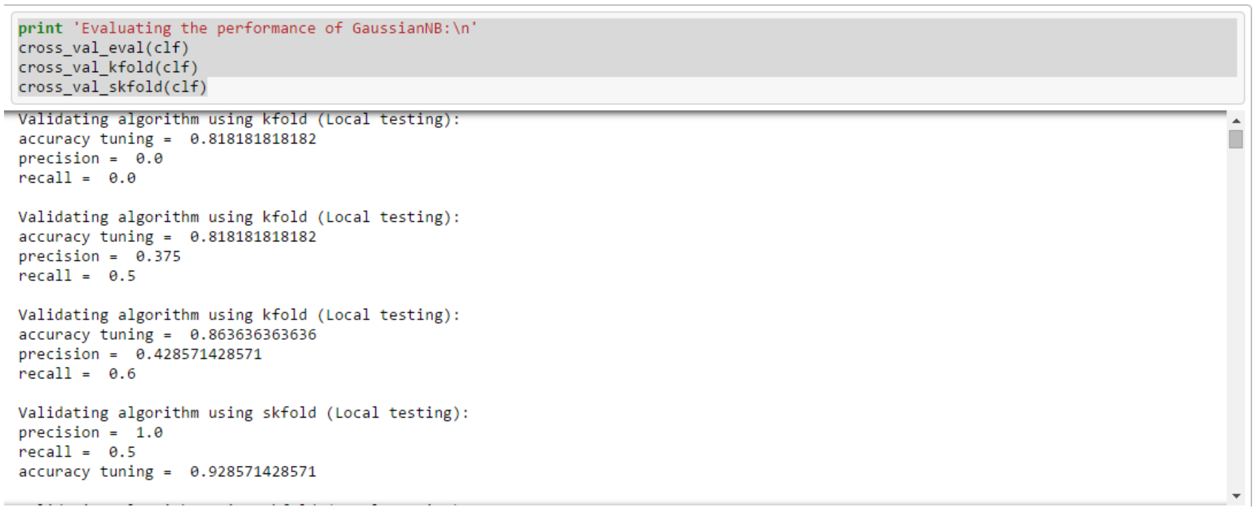
```
def cross_val_kfold(clf):
    ### use KFold for split and validate algorithm
    from sklearn.cross_validation import KFold
    from sklearn.feature_extraction.text import TfidfVectorizer
    kf=KFold(len(labels),3)
  #  print len(labels)
  #  print kf
    for train_indices, test_indices in kf:
        #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]

        scaler = preprocessing.MinMaxScaler().fit(features_train)
        features_train_transformed = scaler.transform(features_train)
        clf = clf.fit(features_train_transformed,labels_train)
        features_test_transformed = scaler.transform(features_test)
        clf.fit(features_train_transformed, labels_train)
        predicted = clf.predict(features_test_transformed)
        print "\nValidating algorithm using kfold (Local testing):"
        print "accuracy tuning = ",accuracy_score (predicted,labels_test)
        print 'precision = ', precision_score(labels_test,predicted)
        print 'recall = ', recall_score(labels_test,predicted)
```



```
def cross_val_skfold(clf):
    from sklearn.cross_validation import StratifiedShuffleSplit
    ### use SKFold for split and validate algorithm
    cv = StratifiedShuffleSplit(labels, 1000, random_state = 42)
    true_negatives = 0
    false_negatives = 0
    true_positives = 0
    false_positives = 0
    for train_idx, test_idx in cv:
        features_train = []
        features_test  = []
        labels_train   = []
        labels_test    = []
        for ii in train_idx:
            features_train.append( features[ii] )
            labels_train.append( labels[ii] )
        for jj in test_idx:
            features_test.append( features[jj] )
            labels_test.append( labels[jj] )

        scaler = preprocessing.MinMaxScaler().fit(features_train)
        features_train_transformed = scaler.transform(features_train)
        clf = clf.fit(features_train_transformed, labels_train)
        features_test_transformed = scaler.transform(features_test)
        predicted= clf.predict(features_test_transformed)
        print "\nValidating algorithm using skfold (Local testing):"
        print 'precision = ', precision_score(labels_test,predicted)
        print 'recall = ', recall_score(labels_test,predicted)
        print "accuracy tuning = ", accuracy_score(labels_test, predicted)
```

In Stratified Shuffle split, during my local testing I iterated 1000 times to see if the recall and precision for the different sets were consistent and picked the algorithm based on review of those results. But for submission purpose even though it will iterate 1000 times it will only output set of scores for the last iteration.

I.e. Below is how the results looked during my testing (notice scroll bar*)



```
print 'Evaluating the performance of GaussianNB:\n'
cross_val_eval(clf)
cross_val_kfold(clf)
cross_val_skfold(clf)
```

```
Validating algorithm using kfold (Local testing):
accuracy tuning =  0.818181818182
precision =  0.0
recall =  0.0

Validating algorithm using kfold (Local testing):
accuracy tuning =  0.818181818182
precision =  0.375
recall =  0.5

Validating algorithm using kfold (Local testing):
accuracy tuning =  0.863636363636
precision =  0.428571428571
recall =  0.6

Validating algorithm using skfold (Local testing):
precision =  1.0
recall =  0.5
accuracy tuning =  0.928571428571
```

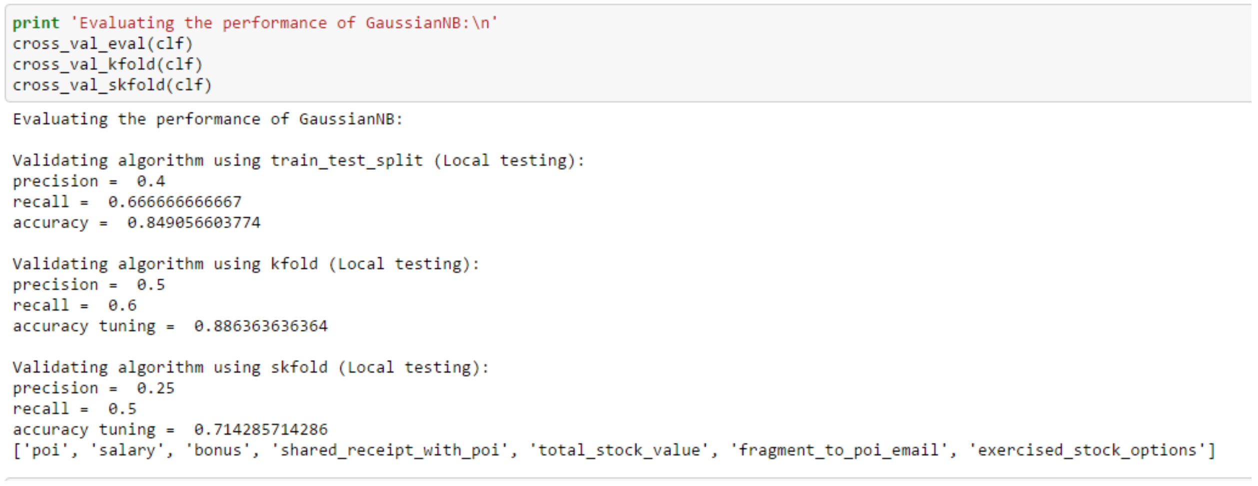
```
print 'Evaluating the performance of KNeighborsClassifier:\n'
cross_val_eval(clf)
cross_val_skfold(clf)
cross_val_kfold(clf)
```

```
Validating algorithm using skfold (Local testing):
precision =  0.5
recall =  0.5
accuracy tuning =  0.857142857143

Validating algorithm using kfold (Local testing):
accuracy tuning =  0.863636363636
precision =  1.0
recall =  0.142857142857

Validating algorithm using kfold (Local testing):
accuracy tuning =  0.886363636364
precision =  1.0
recall =  0.166666666667

Validating algorithm using kfold (Local testing):
accuracy tuning =  0.863636363636
precision =  0.0
recall =  0.0
```

```
print 'Evaluating the performance DecisionTreeClassifier:\n'
cross_val_eval(clf)
cross_val_skfold(clf)
cross_val_kfold(clf)
```

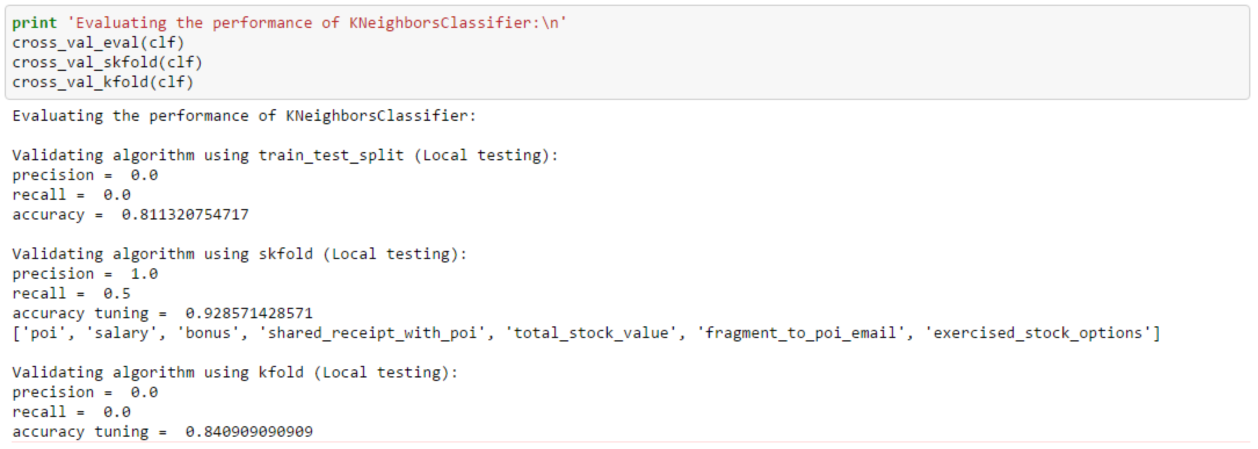
```
Validating algorithm using skfold (Local testing):
precision =  0.666666666667
recall =  1.0
accuracy tuning =  0.928571428571

Validating algorithm using kfold (Local testing):
accuracy tuning =  0.795454545455
precision =  0.375
recall =  0.428571428571

Validating algorithm using kfold (Local testing):
accuracy tuning =  0.795454545455
precision =  0.363636363636
recall =  0.666666666667

Validating algorithm using kfold (Local testing):
accuracy tuning =  0.840909090909
precision =  0.333333333333
recall =  0.4
```

Here are the results of cross validation before tuning



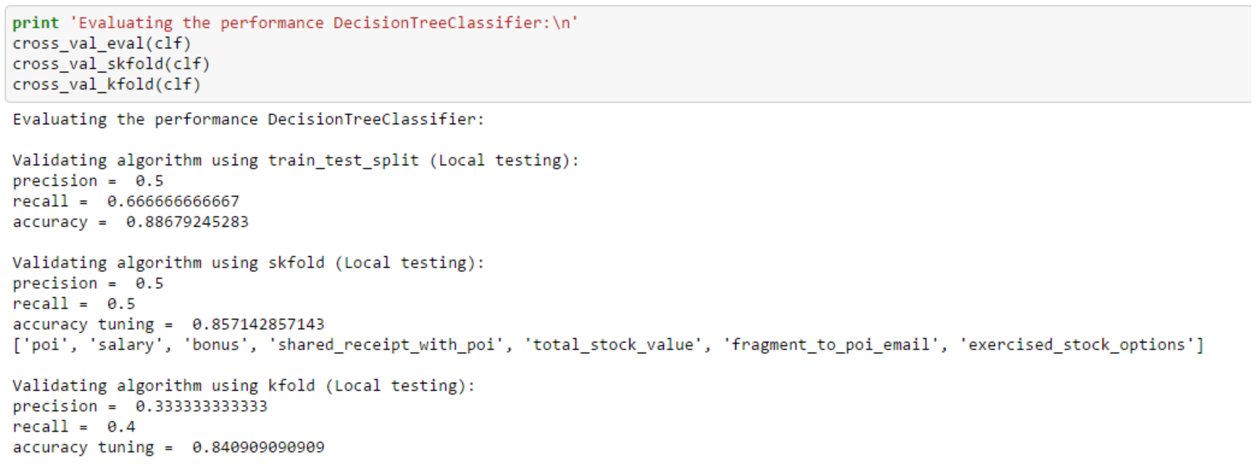
```
print 'Evaluating the performance of GaussianNB:\n'
cross_val_eval(clf)
cross_val_kfold(clf)
cross_val_skfold(clf)
```

```
Evaluating the performance of GaussianNB:

Validating algorithm using train_test_split (Local testing):
precision =  0.4
recall =  0.666666666667
accuracy =  0.849056603774

Validating algorithm using kfold (Local testing):
precision =  0.5
recall =  0.6
accuracy tuning =  0.886363636364

Validating algorithm using skfold (Local testing):
precision =  0.25
recall =  0.5
accuracy tuning =  0.714285714286
['poi', 'salary', 'bonus', 'shared_receipt_with_poi', 'total_stock_value', 'fragment_to_poi_email', 'exercised_stock_options']
```



```
print 'Evaluating the performance of KNeighborsClassifier:\n'
cross_val_eval(clf)
cross_val_skfold(clf)
cross_val_kfold(clf)
```

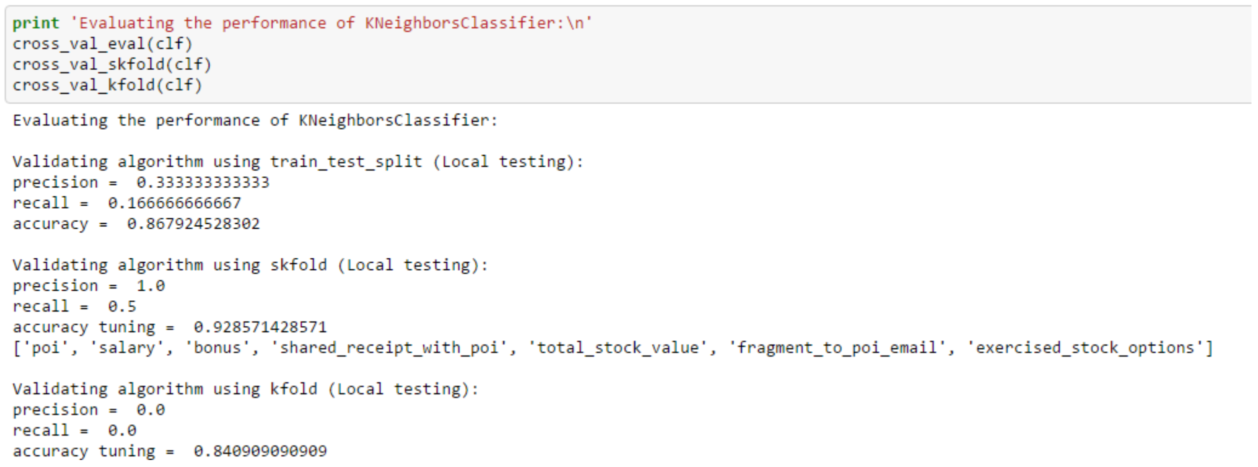
```
Evaluating the performance of KNeighborsClassifier:

Validating algorithm using train_test_split (Local testing):
precision =  0.0
recall =  0.0
accuracy =  0.811320754717

Validating algorithm using skfold (Local testing):
precision =  1.0
recall =  0.5
accuracy tuning =  0.928571428571
['poi', 'salary', 'bonus', 'shared_receipt_with_poi', 'total_stock_value', 'fragment_to_poi_email', 'exercised_stock_options']

Validating algorithm using kfold (Local testing):
precision =  0.0
recall =  0.0
accuracy tuning =  0.840909090909
```



```
print 'Evaluating the performance DecisionTreeClassifier:\n'
cross_val_eval(clf)
cross_val_skfold(clf)
cross_val_kfold(clf)
```

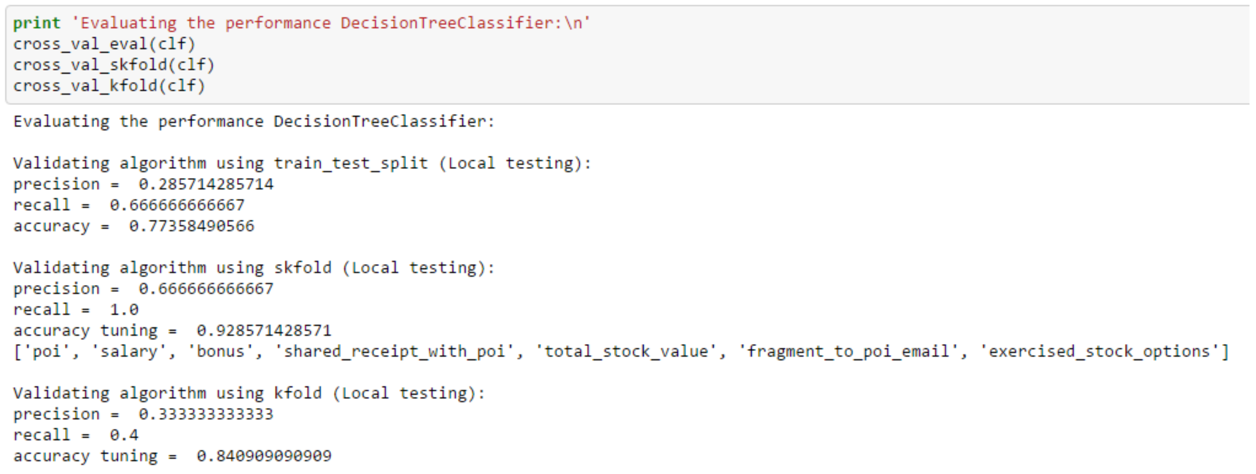
```
Evaluating the performance DecisionTreeClassifier:

Validating algorithm using train_test_split (Local testing):
precision =  0.5
recall =  0.666666666667
accuracy =  0.88679245283

Validating algorithm using skfold (Local testing):
precision =  0.5
recall =  0.5
accuracy tuning =  0.857142857143
['poi', 'salary', 'bonus', 'shared_receipt_with_poi', 'total_stock_value', 'fragment_to_poi_email', 'exercised_stock_options']

Validating algorithm using kfold (Local testing):
precision =  0.333333333333
recall =  0.4
accuracy tuning =  0.840909090909
```

After tuning of parameters the cross validation results were.



```
print 'Evaluating the performance of KNeighborsClassifier:\n'
cross_val_eval(clf)
cross_val_skfold(clf)
cross_val_kfold(clf)
```

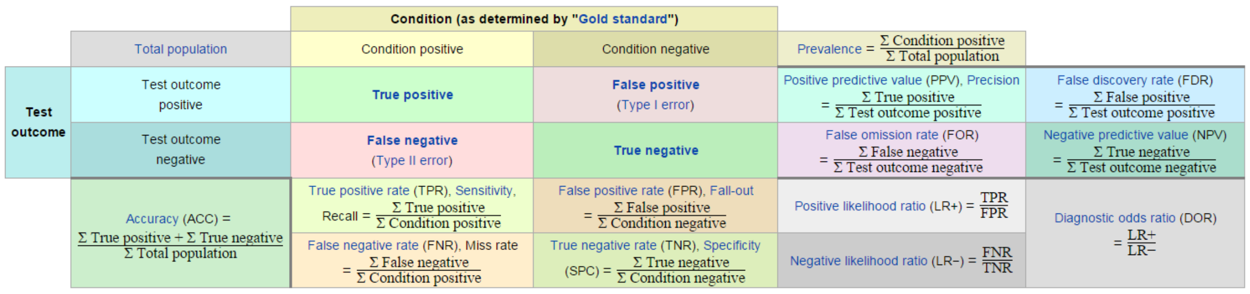
```
Evaluating the performance of KNeighborsClassifier:

Validating algorithm using train_test_split (Local testing):
precision =  0.333333333333
recall =  0.166666666667
accuracy =  0.867924528302

Validating algorithm using skfold (Local testing):
precision =  1.0
recall =  0.5
accuracy tuning =  0.928571428571
['poi', 'salary', 'bonus', 'shared_receipt_with_poi', 'total_stock_value', 'fragment_to_poi_email', 'exercised_stock_options']

Validating algorithm using kfold (Local testing):
precision =  0.0
recall =  0.0
accuracy tuning =  0.840909090909
```



```
print 'Evaluating the performance DecisionTreeClassifier:\n'
cross_val_eval(clf)
cross_val_skfold(clf)
cross_val_kfold(clf)
```

```
Evaluating the performance DecisionTreeClassifier:

Validating algorithm using train_test_split (Local testing):
precision =  0.285714285714
recall =  0.666666666667
accuracy =  0.77358490566

Validating algorithm using skfold (Local testing):
precision =  0.666666666667
recall =  1.0
accuracy tuning =  0.928571428571
['poi', 'salary', 'bonus', 'shared_receipt_with_poi', 'total_stock_value', 'fragment_to_poi_email', 'exercised_stock_options']

Validating algorithm using kfold (Local testing):
precision =  0.333333333333
recall =  0.4
accuracy tuning =  0.840909090909
```

6. Give at least 2 evaluation metrics and your average performance for each of them. Explain an interpretation of your metrics that says something human-understandable about your algorithm's performance. [relevant rubric item: "usage of evaluation metrics"]



		Condition (as determined by "Gold standard")			
	Total population	Condition positive	Condition negative	Prevalence = $\frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	
Test outcome	Test outcome positive	**True positive**	**False positive** (Type I error)	Positive predictive value (PPV), Precision $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Test outcome positive}}$	False discovery rate (FDR) $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Test outcome positive}}$
	Test outcome negative	**False negative** (Type II error)	**True negative**	False omission rate (FOR) $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Test outcome negative}}$	Negative predictive value (NPV) $= \frac{\Sigma \text{ True negative}}{\Sigma \text{ Test outcome negative}}$
	Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$	True positive rate (TPR), Sensitivity, Recall = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{TPR}{FPR}$	Diagnostic odds ratio (DOR) $= \frac{LR+}{LR-}$
		False negative rate (FNR), Miss rate $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	True negative rate (TNR), Specificity (SPC) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR−) = $\frac{FNR}{TNR}$	

Precision is (TP)/ (TP + FP) → which tells us what proportion of POI's we identified as being involved in fraud are actually involved in fraud, i.e. the proportion of TP (true positive) in the set of positive identification of POI's.

Recall is (TP)/(TP + FN) → which tells us what proportion of POI's who were actually involved in fraud were identified by us as POI's, i.e. proportion of TP (true positive) in the set of true POI's.

I.e., Recall gives us information about a classifier's performance with respect to false negatives (how many did we miss), while precision gives us information about its performance with respect to false positives. In simple terms, high precision means that an algorithm returned substantially more relevant results than irrelevant, while high recall means that an algorithm returned most of the relevant results.

Using DecisionTree classifier both precision and recall are >0.3

```
clf = DecisionTreeClassifier(criterion='entropy',random_state=42)
```

```
DecisionTreeClassifier(compute_importances=None, criterion=entropy,
            max_depth=None, max_features=None, min_density=None,
            min_samples_leaf=1, min_samples_split=2, random_state=42,
            splitter=best)
        Accuracy: 0.84693       Precision: 0.46282      Recall: 0.44500 F1: 0.45373     F2: 0.44845
        Total predictions: 14000        True positives:  890    False positives: 1033   False negatives: 1110   True negatives: 10967
```

Even though the precision is high and the recall is > 0.3, I did not pick KNeighbors because the results of the cross validation was not consistent for KNeighbors as it was for the DecisionTree classifier, so I picked the DecisionTree classifier.

```
parameters = {'n_neighbors':[5,6,7,8],'weights' : ('distance', 'uniform') }
svr = KNeighborsClassifier(algorithm = 'brute')
clf = GridSearchCV(svr, parameters,scoring = 'recall')
```

```
GridSearchCV(cv=None,
       estimator=KNeighborsClassifier(algorithm=brute, leaf_size=30, metric=minkowski,
           n_neighbors=5, p=2, weights=uniform),
       estimator__algorithm=brute, estimator__leaf_size=30,
       estimator__metric=minkowski, estimator__n_neighbors=5,
       estimator__p=2, estimator__weights=uniform, fit_params={}, iid=True,
       loss_func=None, n_jobs=1,
       param_grid={'n_neighbors': [5, 6, 7, 8], 'weights': ('distance', 'uniform')},
       pre_dispatch=2*n_jobs, refit=True, score_func=None, scoring=recall,
       verbose=0)
        Accuracy: 0.88614       Precision: 0.68190      Recall: 0.38050 F1: 0.48845     F2: 0.41740
        Total predictions: 14000        True positives:  761    False positives:  355   False negatives: 1239   True negatives: 11645
```