The objective of this project is to predict the employees of Enron who may have committed fraud based on the information Enron financial and email datasets. This can be achieved using different machine learning tools. This was one of the high profile corporate frauds in US history.

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”]

   Dataset Details:
   The dataset consists of 146 users details, 18 of them are considered POI, whereas remaining 128 as non-POI. We can use machine learning techniques to understand if we can “identity any patterns in the emails exchanged”. We can understand the relationship between employees salary and their bonuses using regression, and it can be identified whether the members is in board of directors or just an employee.

   The dataset are classified in 3 different sets, financial features, email and POI/non-POI. It comprises of total 21 features.

   financial_features = ['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']

   email_features = ['to_messages', 'email_address', 'from_poi_to_this_person', 'from_messages', 'from_this_person_to_poi', 'shared_receipt_with_poi']

   POI_label = ['poi']

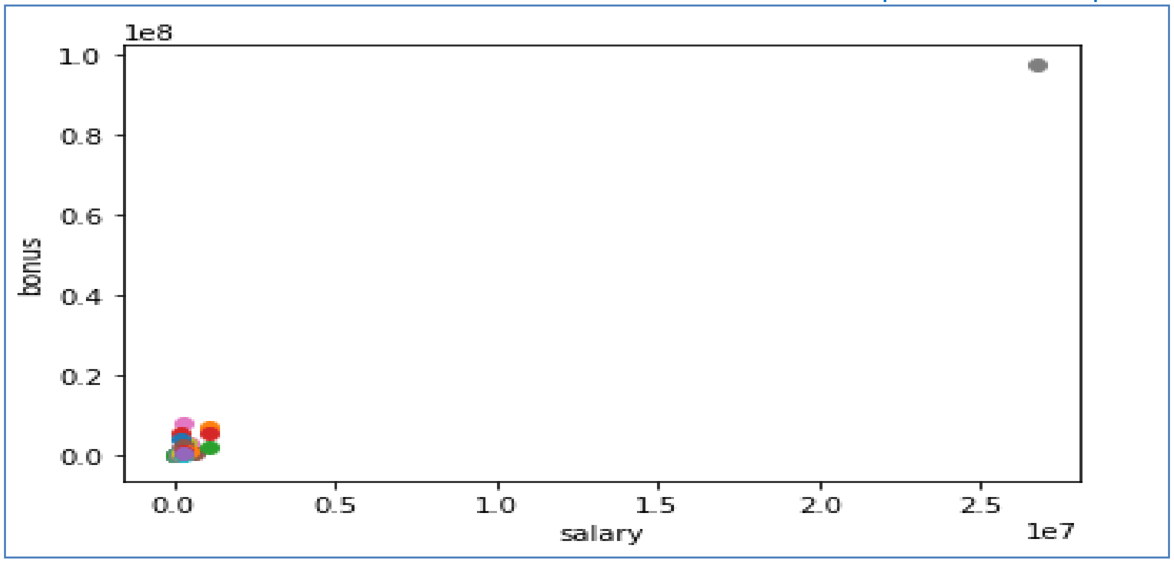
   Few features have many missing values as shown below:

   missing values: {'salary': 51, 'to_messages': 60, 'deferral_payments': 107, 'total_payments': 21, 'exercised_stock_options': 44, 'bonus': 64, 'director_fees': 129, 'restricted_stock_deferred': 128, 'from_messages': 60, 'total_stock_value': 20, 'expenses': 51, 'from_poi_to_this_person': 60, 'loan_advances': 142, 'email_address': 35, 'other': 53, 'from_this_person_to_poi': 60, 'poi': 0, 'deferred_income': 97, 'shared_receipt_with_poi': 60, 'restricted_stock': 36, 'long_term_incentive': 80}

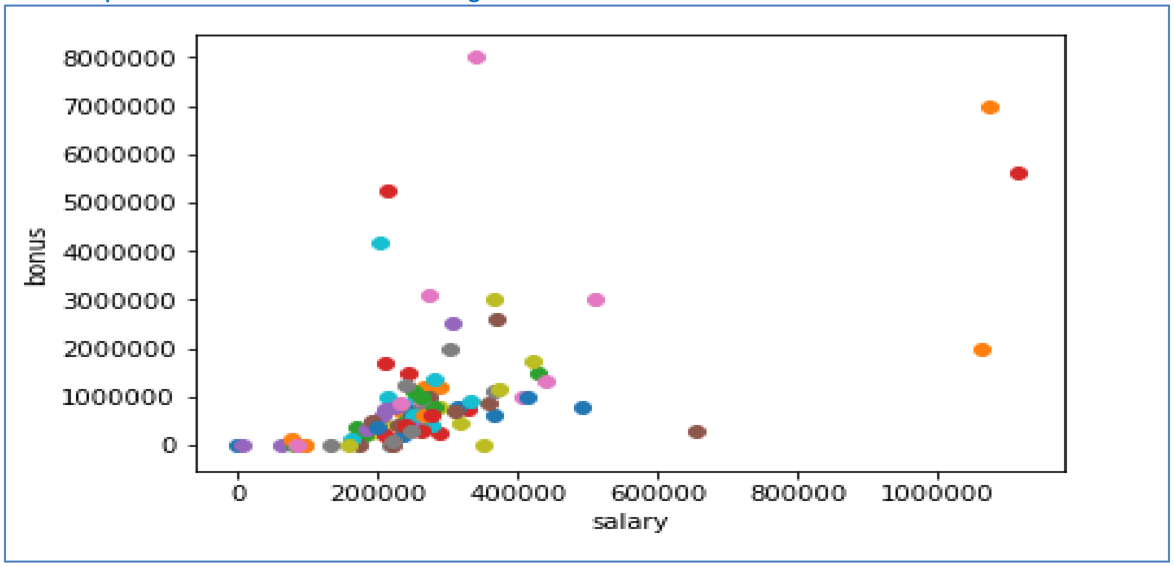
   Missing values are replaced with 0s.

Outliers in the data

One outlier was spotted when the graph of bonus vs salary was plotted. This outlier was 'Total' column. It was removed as it is spreadsheet quirk.



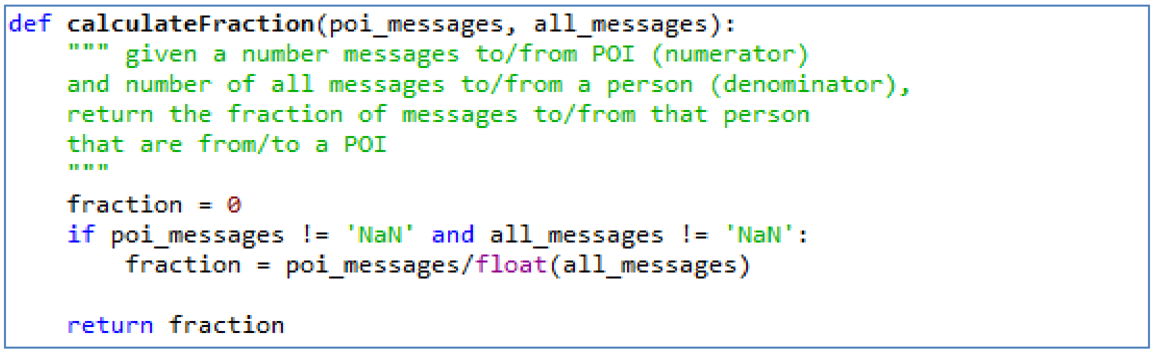
Scatter plot was clear after removing the outlier. It was skewed earlier due to the outlier.



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

   Created 2 new features **fraction_from_poi** and **fraction_to_poi,** which represents the ratio of messages from POI to this person against all messages to this person and ratio of messages from this person to POI against all messages from this person respectively. Then added these 2 new features to the features list and passed it to SelectKBest function for feature selection.

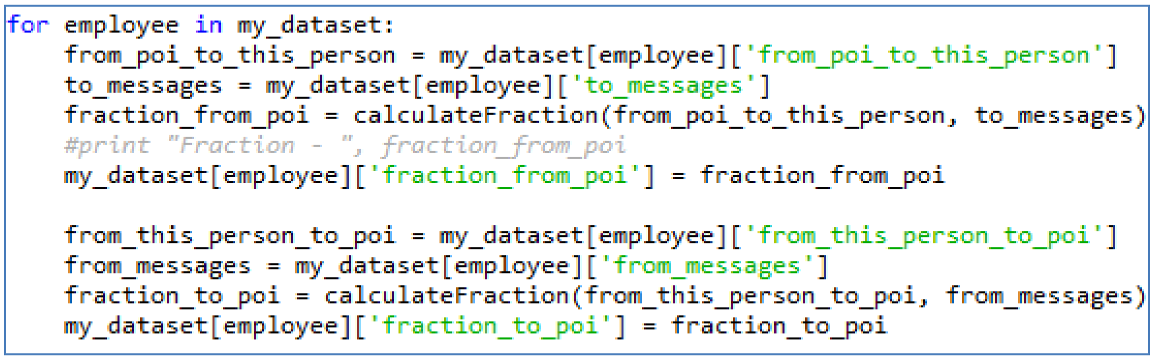
   The below code does the calculation of messages to/from that person that are from/to a POI, given a number of messages to/from POI (numerator) and number of all messages to/from a person (denominator).



```
def calculateFraction(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
    """
    fraction = 0
    if poi_messages != 'NaN' and all_messages != 'NaN':
        fraction = poi_messages/float(all_messages)

    return fraction
```

   Used the above function to calculate the fraction of employee in the data set and added the new features of the original dataset.



```
for employee in my_dataset:
    from_poi_to_this_person = my_dataset[employee]['from_poi_to_this_person']
    to_messages = my_dataset[employee]['to_messages']
    fraction_from_poi = calculateFraction(from_poi_to_this_person, to_messages)
    #print "Fraction - ", fraction_from_poi
    my_dataset[employee]['fraction_from_poi'] = fraction_from_poi

    from_this_person_to_poi = my_dataset[employee]['from_this_person_to_poi']
    from_messages = my_dataset[employee]['from_messages']
    fraction_to_poi = calculateFraction(from_this_person_to_poi, from_messages)
    my_dataset[employee]['fraction_to_poi'] = fraction_to_poi
```

Univariate Feature Selection

To decide and conclude the best feature to use, I used the SelectKBest feature selection function, which selects the K features which are most powerful (K is the parameter). Below are the returned values from the function:

```
[('exercised_stock_options', 25.097541528735491),
('total_stock_value', 24.467654047526398),
('bonus', 21.060001707536571),
('salary', 18.575703268041785),
('fraction_to_poi', 16.641707070468989),
('deferred_income', 11.595547659730601),
('long_term_incentive', 10.072454529369441),
('restricted_stock', 9.3467007910514877),
('total_payments', 8.8667215371077717),
('shared_receipt_with_poi', 8.7464855321290802),
('loan_advances', 7.2427303965360181),
('expenses', 6.2342011405067401),
('from_poi_to_this_person', 5.3449415231473374),
('other', 4.204970858301416),
('fraction_from_poi', 3.2107619169667441),
('from_this_person_to_poi', 2.4265081272428781),
('director_fees', 2.1076559432760908),
('to_messages', 1.6988243485808501),
('deferral_payments', 0.2170589303395084),
('from_messages', 0.16416449823428736),
('restricted_stock_deferred', 0.06498431172371151)]
```

To observe the effect of new features added, I looked at the performance of each algorithm with original best features and compared it with performance after adding the new features:

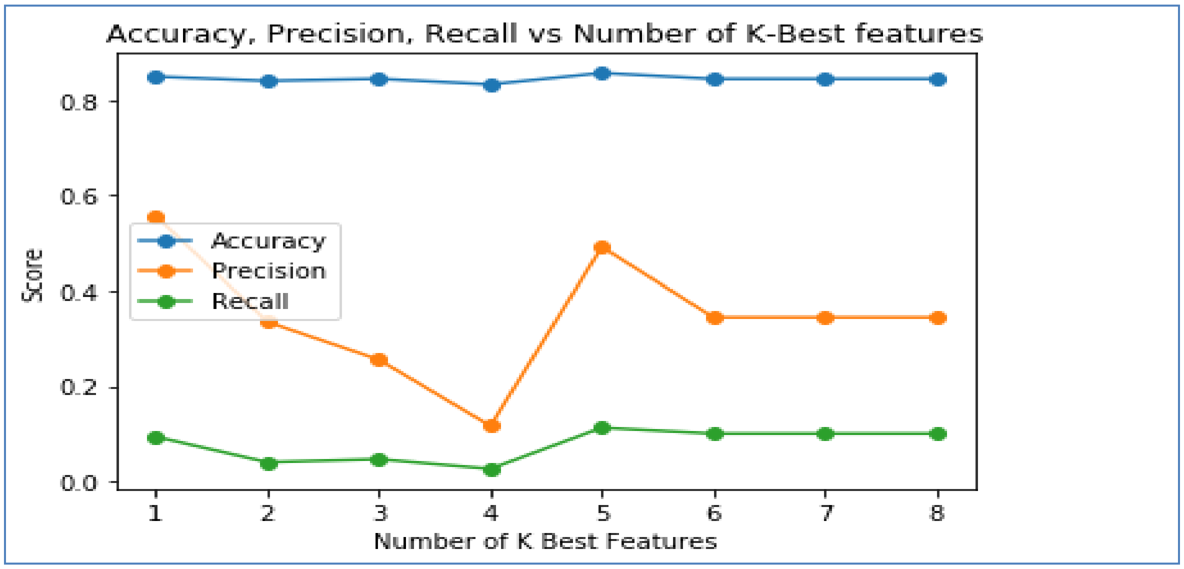
Algorithm	**Precision**	**Recall**	**Precision**	**Recall**
	Original Features	**Original Features**	**New Features**	**New Features**
Naive Bayes	0.44523989899	0.391672979798	0.401189123377	0.392877886003
SVM	0.146388888889	0.0502827380952	0.226458333333	0.096994047619
Decision Tree	0.25405799062	0.26775974026	0.288734321234	0.280880681818
Random Forest	0.330625	0.148264339827	0.388125	0.192481060606
Logistic Regression	0.0	0.0	0.0	0.0

It can be noticed that the performance differs from one algorithm to another. Some algorithms perform better while including the new features, e.g. SVM and Decision Tree. Some algorithms perform worse with the new features included, e.g. Naive Bayes and Random Forest.

I decided to select 8 best features along with POI as they obtained the highest scores from SelectKBest results. The feature list is mentioned below:

```
KBest ['poi', 'exercised_stock_options', 'total_stock_value',
'bonus', 'salary', 'deferred_income', 'long_term_incentive',
'restricted_stock']
```

Below graph plots and shows the different scores of accuracy, precision & recall against the different values of K.



Feature Scaling

The selected features had different units and some had very big values, it needed to be transformed. I used MinMaxScaler from sklearn to scale all the selected features in the scale between 0 and 1.

```
from sklearn import preprocessing
data = featureFormat(my_dataset, kBest_features, sort_keys = True)
labels, features = targetFeatureSplit(data)
scaler = preprocessing.MinMaxScaler()
features = scaler.fit_transform(features)
```

3. 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 tried 5 different algorithms as mentioned below and ended up using Naïve Bayes as its evaluation score is highest amongst all.

   Other tried algorithms are as below:

   - SVM
   - Decision Tree
   - Random Forest
   - Logistic Regression

I observed that, SVM and Logistic Regression took longer to run. Other observation was that all algorithms showed high accuracy, which indicates that this is not the best evaluation metric. Also, because it’s not ideal for skewed datasets.

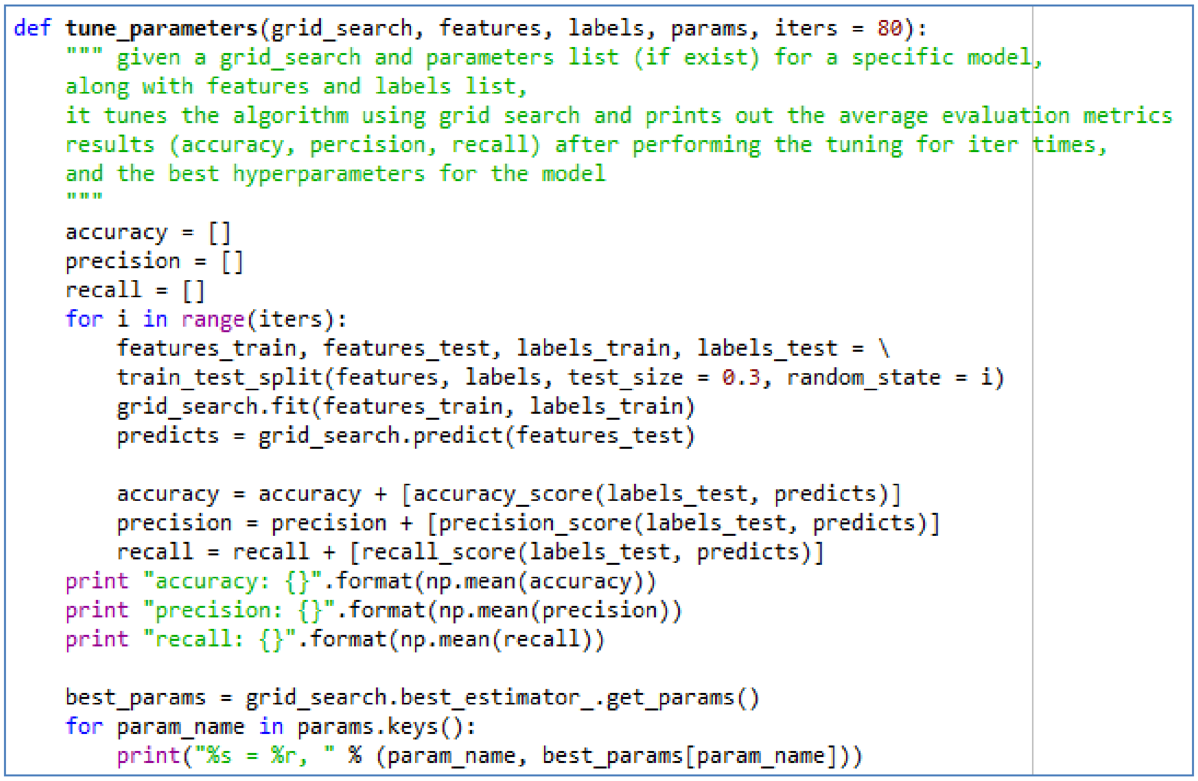
4. What does it mean to tune the parameters of an algorithm, and what can happen if you don’t do this well? How did you tune the parameters of your particular algorithm? What parameters did you tune? (Some algorithms do not have parameters that you need to tune -- if this is the case for the one you picked, identify and briefly explain how you would have done it for the model that was not your final choice or a different model that does utilize parameter tuning, e.g. a decision tree classifier). [relevant rubric items: “discuss parameter tuning”, “tune the algorithm”]

   Classifiers come with parameters with default values which impact the model. Explicit values can be passed as parameter to constructors as well. Most common examples are “C”, “kernel”, and “gamma” for SVM classifier. Classifiers parameters are tuned to optimize the parameters so that the algorithms perform its best. It should be done before presenting the final results. If this part is skipped then it can be miss-fitting the data in the model.

   Algorithm Tuning:

   I used GridSearchCV from sklearn for parameter tuning in algorithms SVM, Decision Tree and Logisitic regression.

   I defined the below function to tune the algorithm using grid search. It prints the best hyperparamters for all the models after performing tuning for 80 iterations, along with evaluation metrics results of accuracy, precision & recall.

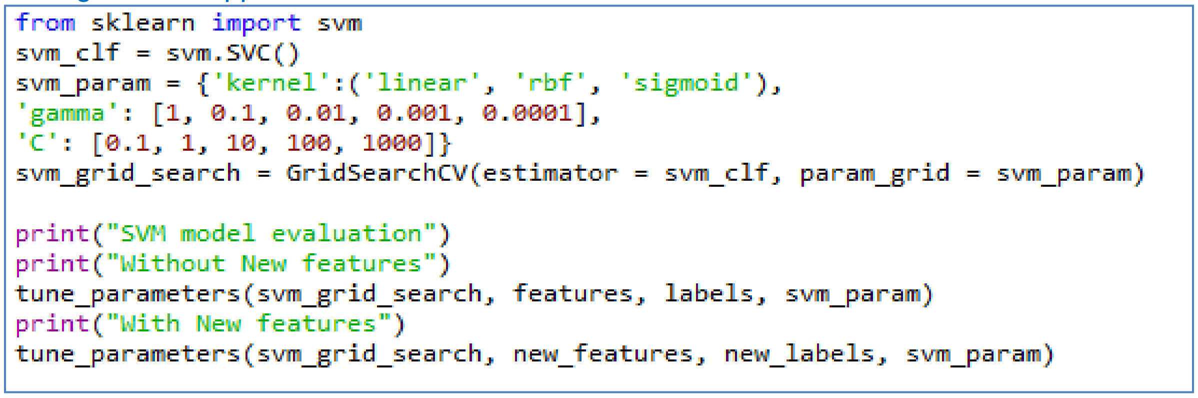


```python
def tune_parameters(grid_search, features, labels, params, iters = 80):
    """ given a grid_search and parameters list (if exist) for a specific model,
    along with features and labels list,
    it tunes the algorithm using grid search and prints out the average evaluation metrics
    results (accuracy, percision, recall) after performing the tuning for iter times,
    and the best hyperparameters for the model
    """
    accuracy = []
    precision = []
    recall = []
    for i in range(iters):
        features_train, features_test, labels_train, labels_test = \
        train_test_split(features, labels, test_size = 0.3, random_state = i)
        grid_search.fit(features_train, labels_train)
        predicts = grid_search.predict(features_test)

        accuracy = accuracy + [accuracy_score(labels_test, predicts)]
        precision = precision + [precision_score(labels_test, predicts)]
        recall = recall + [recall_score(labels_test, predicts)]
    print "accuracy: {}".format(np.mean(accuracy))
    print "precision: {}".format(np.mean(precision))
    print "recall: {}".format(np.mean(recall))

    best_params = grid_search.best_estimator_.get_params()
    for param_name in params.keys():
        print("%s = %r, " % (param_name, best_params[param_name]))
```

Tuning Vector support classification:



```
from sklearn import svm
svm_clf = svm.SVC()
svm_param = {'kernel':('linear', 'rbf', 'sigmoid'),
'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
'C': [0.1, 1, 10, 100, 1000]}
svm_grid_search = GridSearchCV(estimator = svm_clf, param_grid = svm_param)

print("SVM model evaluation")
print("Without New features")
tune_parameters(svm_grid_search, features, labels, svm_param)
print("With New features")
tune_parameters(svm_grid_search, new_features, new_labels, svm_param)
```

Result

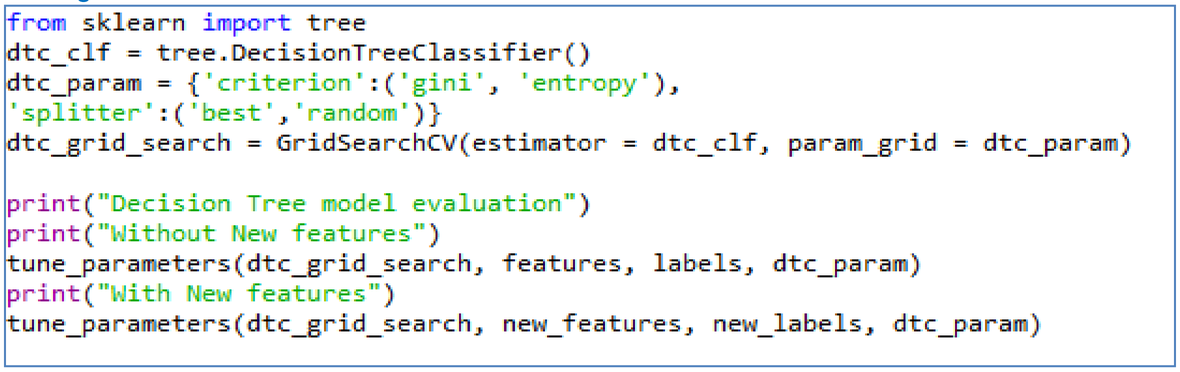
Without new features:

kernel = 'linear',
C = 1,
gamma = 1,

With new features:

kernel = 'linear',
C = 0.1,
gamma = 1,

Tuning Decision Tree Classification



```
from sklearn import tree
dtc_clf = tree.DecisionTreeClassifier()
dtc_param = {'criterion':('gini', 'entropy'),
'splitter':('best','random')}
dtc_grid_search = GridSearchCV(estimator = dtc_clf, param_grid = dtc_param)

print("Decision Tree model evaluation")
print("Without New features")
tune_parameters(dtc_grid_search, features, labels, dtc_param)
print("With New features")
tune_parameters(dtc_grid_search, new_features, new_labels, dtc_param)
```

Result

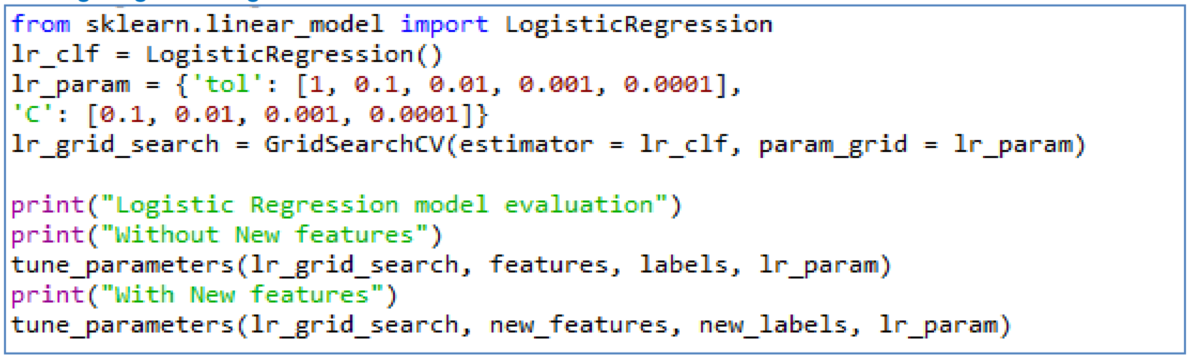
Without new features:

splitter = ‘best’, criterion = 'gini

With new features:

splitter = ''random', criterion = 'gini',

Tuning Logistic Regression Classification:



```
from sklearn.linear_model import LogisticRegression
lr_clf = LogisticRegression()
lr_param = {'tol': [1, 0.1, 0.01, 0.001, 0.0001],
'C': [0.1, 0.01, 0.001, 0.0001]}
lr_grid_search = GridSearchCV(estimator = lr_clf, param_grid = lr_param)

print("Logistic Regression model evaluation")
print("Without New features")
tune_parameters(lr_grid_search, features, labels, lr_param)
print("With New features")
tune_parameters(lr_grid_search, new_features, new_labels, lr_param)
```

Result

Without new features:

C = 0.1,
tol = 1,

With new features:

C = 0.1,
tol = 1,

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

   Validation is process of testing our algorithm. In validation of the algorithm, we compare the results of the test data with training data. Benefit of having the training and tests data, it allows us to check and ensure that we are not over –fitting the data. It also provides a way to check if we are getting the expected results.

The most common mistake in validation process is not splitting the data in training/testing datasets, which results in over fitting of data. I used train_test_split to split 30% of my data to testing, then used sklearn's accuracy, precision & recall scores to validate the algorithms.

The table below shows the results for the chosen algorithms.

Algorithm	**Accuracy**	**Precision**	**Recall**
Naive Bayes	0.849431818182	0.44523989899	0.391672979798
SVM	0.868465909091	0.146388888889	0.0502827380952
Decision Tree	0.810511363636	0.25405799062	0.26775974026
Random Forest	0.330625	0.148264339827	0.150128968254
Logistic Regression	0.871726190476	0.0	0.0

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"]

Recall: Recall is also referred to as the true positive rate or sensitivity. In the current context it means, the proportion of the poi the model can detect out of all the poi. In such context generally higher recall is preferred.
It's calculated as - True Positive / (True Positive + False Negative). Out of all the items that are truly positive, how many were correctly classified as positive.

Precision: Precision is also referred as truly predicted value. In current context it means, the proportion of the correct prediction of all the people who are predicted to be poi.
It is calculated as - True Positive / (True Positive + False Positive). Out of all the items labeled as positive, how many truly belong to the positive class.

The chosen algorithm is Naive Bayes, which resulted in precision of 0.44523989899 and recall of 0.391672979798

References:

1. https://docs.google.com/document/d/1NDgi1PrNJP7WTbfSUuRUnz8yzs5nGVTSzpO7oeNTEWA/pub?embedded=true
2. https://github.com/evykassirer/udacity-machine-learning
3. http://scikit-learn.org/stable/modules/model_evaluation.html
4. http://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html
5. https://discussions.udacity.com/t/project-fear-strugging-with-machine-learning-project/198529
6. https://github.com/jasminej90/dand5-identity-fraud-from-enron-email
7. http://ghunt03.github.io/DAProjects/DAP05/report.html