# Enron POI Classifier Model Building

April 16, 2017

## 1 Enron POI Classifier Model Building

### 1.1 Initializing and checking versions

```
In [1]: %cd C:\Users\damia\OneDrive\Dokumente\Udacity\DataAnalyst\EnronFinal\final_
        import sys
        import sklearn
        print "sklearn", sklearn.__version__
        print "python", sys.version
```

```
C:\Users\damia\OneDrive\Dokumente\Udacity\DataAnalyst\EnronFinal\final_project
sklearn 0.17.1
python 2.7.12 |Anaconda 4.1.1 (64-bit)| (default, Jun 29 2016, 11:07:13) [MSC v.150
```

### 1.2 Loading the cleaned data (see EDA document for details)

```
In [2]: import sys
        import pickle
        import numpy as np
        with open("clean_data.pkl", "r") as data_file:
            data_dict = pickle.load(data_file)
```

#### 1.2.1 Collect labels and features in appropriate arrays

```
In [14]: def init_feature_names():
             global feature_names
             feature_names = sorted(data_dict['ALLEN PHILLIP K'].keys())
             feature_names.remove('poi') # the label
             feature_names.remove('email_address') # unique for each data point, th
             feature_names.remove('total_payments') # linear combination of other f
             feature_names.remove('total_stock_value') # linear combination of othe
         def update_data():
             global labels, features
             labels = [1.0 if d['poi'] else 0.0 for d in data_dict.values()]
             features = [[float(d[f]) if f in d and d[f] != 'NaN' else 0.0 for f in
         init_feature_names()
         update_data()
```

### 1.2.2 Standardize features for use in models that are sensitive to magnitudes

In [6]:
```
def get_feature_values(fidx):
    return [features[i][fidx] for i in range(len(features))]
mu_f = [np.mean(get_feature_values(fidx)) for fidx in range(len(feature_nam
sd_f = [np.std(get_feature_values(fidx)) for fidx in range(len(feature_name
norm_features = []
for idx in range(len(features)):
    data = []
    for fidx in range(len(feature_names)):
        data.append((features[idx][fidx] - mu_f[fidx]) / sd_f[fidx] if sd_f
    norm_features.append(data)
```

### 1.2.3 Import various stuff needed for model building

In [8]:
```
from sklearn.cross_validation import StratifiedShuffleSplit
from sklearn.grid_search import GridSearchCV
```

## 1.3 Decision Tree Classifier

### 1.3.1 Feature selection

In [9]:
```
from sklearn.tree import DecisionTreeClassifier
splitter = StratifiedShuffleSplit(labels, n_iter=1000, random_state=42)
scores = {}
for name in feature_names:
    scores[name] = []
for idx_train, idx_test in splitter:
    features_train = [features[i] for i in idx_train]
    features_test = [features[i] for i in idx_test]
    labels_train = [labels[i] for i in idx_train]
    labels_test = [labels[i] for i in idx_test]
    clf = DecisionTreeClassifier()
    clf.fit(features_train, labels_train)
    for i in range(len(feature_names)):
        scores[feature_names[i]].append(clf.feature_importances_[i])
for name, values in scores.items():
    scores[name] = np.mean(values)
cumulative_score = 0
for name, score in sorted(scores.items(), key=lambda x: x[1], reverse=True)
    cumulative_score += score
    print "%-25s: %.3f %.3f" % (name, score, cumulative_score)
```

```
exercised_stock_options  : 0.198 0.198
fraction_to_poi          : 0.149 0.348
other                    : 0.124 0.471
shared_receipt_with_poi  : 0.111 0.583
expenses                 : 0.107 0.690
bonus                    : 0.096 0.786
```

```
restricted_stock          : 0.041 0.827
long_term_incentive       : 0.036 0.863
deferred_income           : 0.036 0.899
salary                    : 0.021 0.920
from_messages             : 0.019 0.940
from_this_person_to_poi   : 0.019 0.958
deferral_payments         : 0.011 0.969
fraction_from_poi         : 0.011 0.980
from_poi_to_this_person   : 0.010 0.989
to_messages               : 0.009 0.998
loan_advances             : 0.001 0.999
restricted_stock_deferred: 0.001 1.000
director_fees             : 0.000 1.000
```

I select the top 5 features for the model

```
In [10]: feature_names = ['exercised_stock_options', 'fraction_to_poi', 'other', 's
         update_data()
```

### 1.3.2 Algorithm Tuning

```
In [12]: parameters = {'max_depth': (1, 2, 3, 4, 5, None),
                       'min_samples_split': (2, 3, 4),
                       'min_samples_leaf': (1, 2, 3, 4),
                       'max_leaf_nodes': (2, 3, 4, 5, 10, None),
                       'criterion': ('gini', 'entropy')
                      }
         clf = GridSearchCV(DecisionTreeClassifier(),
                            parameters,
                            scoring='f1',
                            cv=StratifiedShuffleSplit(labels, n_iter=100, random_st
         clf.fit(features, labels)
         print 'Score: %0.3f' % clf.best_score_
         print 'Parameters:'
         for name, value in sorted(clf.best_estimator_.get_params().items()):
             print '%s: %r' % (name, value)
```

```
Score: 0.513
Parameters:
class_weight: None
criterion: 'entropy'
max_depth: 4
max_features: None
max_leaf_nodes: None
min_samples_leaf: 4
min_samples_split: 3
min_weight_fraction_leaf: 0.0
presort: False
```

```
random_state: None
splitter: 'best'
```

## 1.4 Logistic Regression (Lasso)

### 1.4.1 Feature selection

```
In [32]: init_feature_names()
         update_data()
         from sklearn.linear_model import LogisticRegressionCV, LogisticRegression
         clf = LogisticRegressionCV(cv=StratifiedShuffleSplit(labels, n_iter=1000,
                                    scoring='f1',
                                    solver='liblinear',
                                    penalty='l1')
         clf.fit(norm_features, labels)
         coefs = [(feature_names[idx], clf.coef_[0][idx]) for idx in range(len(feat
         for name, value in sorted(coefs, key=lambda x: abs(x[1]), reverse=True):
             print '%-27s: %8.4f' % (name, value)
```

```
from_messages              : -31.1839
restricted_stock_deferred  :  11.7534
from_this_person_to_poi    :   9.6442
director_fees              :  -4.8050
deferral_payments          :  -3.2756
deferred_income            :  -2.5946
from_poi_to_this_person    :   1.5348
salary                     :   1.1790
bonus                      :  -1.1330
exercised_stock_options    :   0.8050
fraction_from_poi          :  -0.8016
expenses                   :   0.7647
other                      :   0.7273
to_messages                :   0.4015
loan_advances              :  -0.3558
long_term_incentive        :  -0.3376
restricted_stock           :  -0.1765
fraction_to_poi            :   0.0921
shared_receipt_with_poi    :   0.0394
```

We take the 9 top features as they have the largest weights

```
In [33]: feature_names = ['from_messages', 'restricted_stock_deferred', 'from_this_
                          'deferral_payments', 'deferred_income', 'from_poi_to_this
         update_data()
```

### 1.4.2 Algorithm tuning

```
In [35]: parameters = {'solver': ('liblinear',),
                       'penalty': ('l1',),
                       'fit_intercept': (True, False),
                       'C': (.25, .5, .9, 1., 2.),
                       'tol': (1e-5, 1e-4, 1e-3)
                      }
         clf = GridSearchCV(LogisticRegression(),
                            parameters,
                            scoring='f1',
                            cv=StratifiedShuffleSplit(labels, n_iter=100, random_st
         clf.fit(norm_features, labels)
         print 'Score: %0.3f' % clf.best_score_
         print 'Parameters:'
         for name, value in sorted(clf.best_estimator_.get_params().items()):
             print '%s: %r' % (name, value)
```

```
Score: 0.441
Parameters:
C: 0.25
class_weight: None
dual: False
fit_intercept: False
intercept_scaling: 1
max_iter: 100
multi_class: 'ovr'
n_jobs: 1
penalty: 'l1'
random_state: None
solver: 'liblinear'
tol: 1e-05
verbose: 0
warm_start: False
```

## 1.5 Final model selection

As the decision tree has a higher F1 score (0.513) than the logistic regression model (0.441), I choose the decision tree model as the final model. In addition to this, it uses less features.