Table of Contents

Section 1. Naive Bayes Classifier

1.1 Learning outcomes

The following topics were studied and practiced using some practice exercises and applied to the mini projects and the final Enron project

- Supervised VS Unsupervised Classification examples
- Classification VS Regression examples
- Gaussian Naive Bayes algorithm
- Normalizing
- Total Probability
- Bayes rule for classification
- Posterior probabilities
- Linear Decision surfaces
- NB decision boundary in python using 'sklearn'
- Text Learning using Bayes Rule – 'TfidfVectorizer' of 'sklearn'

1.2 Practice mini projects

1.2.1 Use sklearn Naive Bayes classifier to classify the terrain data

Objective:

The objective of this exercise is to apply naive Bayes classifier and visually plot a decision boundary for speed VS QualityOfTerrain of terrain data

Data used:

Made up Speed VS Quality Terrain data

Language:

python

Analysis:

Step 1: Prepare the data

```
random.seed(42)
grade = [random.random() for ii in range(0,n_points)]
bumpy = [random.random() for ii in range(0,n_points)]
error = [random.random() for ii in range(0,n_points)]
y = [round(grade[ii]*bumpy[ii]+0.3+0.1*error[ii]) for ii in range(0,n_points)]
for ii in range(0, len(y)):
    if grade[ii]>0.8 or bumpy[ii]>0.8:
        y[ii] = 1.0
```

Step 2: Split data into train/test sets

```
X = [[gg, ss] for gg, ss in zip(grade, bumpy)]
split = int(0.75*n_points)
X_train = X[0:split]
X_test  = X[split:]
y_train = y[0:split]
y_test  = y[split:]
```

```
grade_sig = [X_train[ii][0] for ii in range(0, len(X_train)) if y_train[ii]==0]
bumpy_sig = [X_train[ii][1] for ii in range(0, len(X_train)) if y_train[ii]==0]
grade_bkg = [X_train[ii][0] for ii in range(0, len(X_train)) if y_train[ii]==1]
bumpy_bkg = [X_train[ii][1] for ii in range(0, len(X_train)) if y_train[ii]==1]
```

Step 3: Build a Naive Bayes classifier, train the classifier and classify data

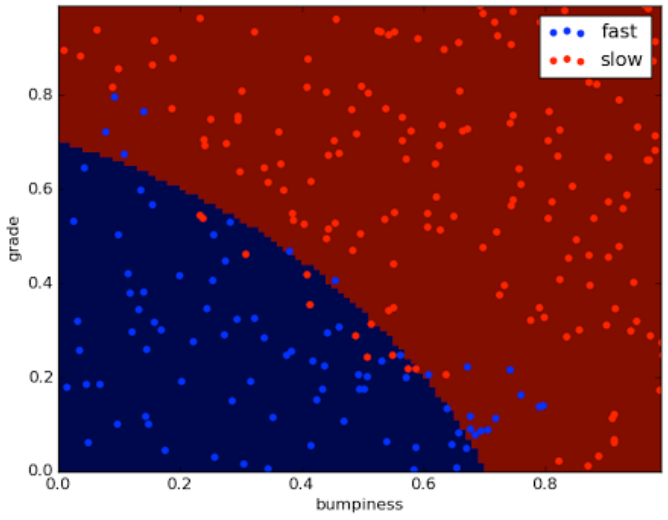
```
def classify(features_train, labels_train):
    ### import the sklearn module for GaussianNB
    ### create classifier
    ### fit the classifier on the training features and labels
    ### return the fit classifier
    from sklearn.naive_bayes import GaussianNB
    clf = GaussianNB()
    clf.fit(features_train, labels_train)
    return clf
```

.....

```
features_train, labels_train, features_test, labels_test = makeTerrainData()

### the training data (features_train, labels_train) have both "fast" and "slow" points mixed
### in together--separate them so we can give them different colors in the scatterplot,
### and visually identify them
grade_fast = [features_train[ii][0] for ii in range(0, len(features_train)) if labels_train[ii]==0]
bumpy_fast = [features_train[ii][1] for ii in range(0, len(features_train)) if labels_train[ii]==0]
grade_slow = [features_train[ii][0] for ii in range(0, len(features_train)) if labels_train[ii]==1]
bumpy_slow = [features_train[ii][1] for ii in range(0, len(features_train)) if labels_train[ii]==1]


clf = classify(features_train, labels_train)
```

Step 4: Draw decision boundary

```
    ### draw the decision boundary with the text points overlaid
prettyPicture(clf, features_test, labels_test)
output_image("test.png", "png", open("test.png", "rb").read())
```



Step 5: Predict on test data and calculate accuracy

```
### use the trained classifier to predict labels for the test features
pred = clf.predict(features_test)

from sklearn.metrics import accuracy_score
accuracy = accuracy_score(labels_test, pred)
```

```
{"accuracy": "0.884"}
```

1.2.2 Text Learning using Naive Bayes – Identify if the author is 'Chris' or 'Sara'

Objective:

Given a set of email texts from two people, Chris and Sara, the objective of this mini-project is to:

- Split the texts into words of training and testing data sets
- Using Text vectorization, convert to tfidf matrix
- Select the most useful words/features
- Finally apply GaussianNB algorithm to predict author based on word

Data used:

Email_text data

Language:

python

Analysis:

Showing the above described process in parts here:

Step 1: We first convert the strings from the email texts into list of numbers by counting the word frequency

```
### text vectorization--convert from strings to lists of numbers (counting word frequency)
vectorizer = TfidfVectorizer(sublinear_tf=True, max_df=0.5,
                             stop_words='english')
features_train_transformed = vectorizer.fit_transform(features_train)
features_test_transformed  = vectorizer.transform(features_test)
```

Step 2: Because, there are too many words, the data can be high-dimensional, so we need to select some distinctive words

```
selector = SelectPercentile(f_classif, percentile=1)
selector.fit(features_train_transformed, labels_train)
features_train_transformed = selector.transform(features_train_transformed).toarray()
features_test_transformed  = selector.transform(features_test_transformed).toarray()
```

Step 3: Train the Gaussian Naive Bayes algorithm on these words and predict the authors for test data

```
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()
#Train
clf.fit(features_train, labels_train)
#Predict on test
pred = clf.predict(features_test)
#calculate Test accuracy
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(labels_test, pred)
```

Results:

```
Kavyas-MacBook-Air:naive_bayes kavyagautam$ python nb_author_id.py
no. of Chris training emails: 7936
no. of Sara training emails: 7884
training time: 1.796 s
predicting time: 0.242 s
TEST ACCURACY: 0.973265073948
Kavyas-MacBook-Air:naive_bayes kavyagautam$
```

----------End of Section 1----------

Section 2. Support Vector Machines

2.1 Learning outcomes

The following topics were studied and practiced using some practice exercises and applied to the mini projects and the final Enron project

- Separating lines
- Margin of the SVM
- Linear Support Vector Machines – Hyperplanes
- SVM decision boundary and outliers
- Non-Linear SVMs
- Adding a new feature to SVM
- The Kernel trick
- Gamma and C parameters of SVM
- Over-fitting
- Speed-Accuracy trade off by optimizing C parameter
- RBF Kernel
- Optimized RBF VS Linear SVM

2.2 Practice mini projects

2.2.1 Use sklearn SVM classifier to classify the terrain data

Objective:

The objective of this exercise is to apply Support Vector classifier and visually plot a decision boundary for speed VS QualityOfTerrain data

Language:
python

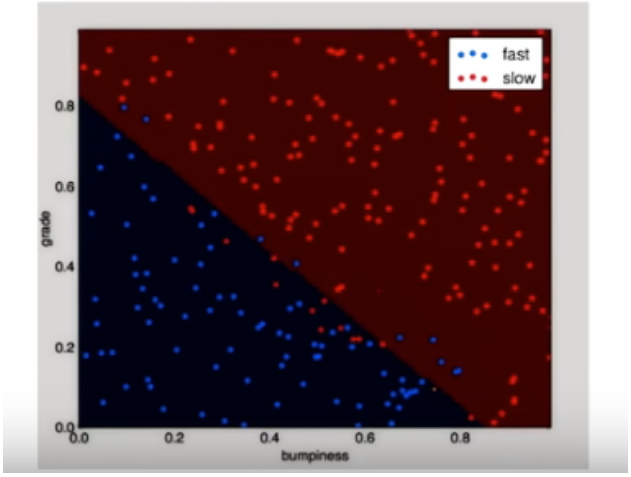
Data used:
Made up Speed VS Quality Terrain data
(same as before)

Analysis:
We follow the same set of steps as before:
- Handle data
- Build a classifier
- Train and then predict on test data

```
from sklearn.svm import SVC
clf = SVC(kernel="linear")
clf.fit(features_train, labels_train)
pred = clf.predict(features_test)

from sklearn.metrics import accuracy_score
acc = accuracy_score(pred, labels_test)
```

Accuracy = 0.92

And since the data is fairly linear, the SVM Linear decision boundary will look like this



However, the data will not always be linear, in such cases, the Kernel trick will come in handy. A kernel is a function of SVM that will transform the data into a high dimensional data by adding new features and convert the data back to 2 dimensions, which will be linearly separable.
So by adjusting the parameters of SVM function – like Kernel = 'rbf' or 'poly' or 'sigmoid' and the C parameter that controls the trade off between "Smooth decision boundary" and "Classifying more training points correctly"

2.2.2 Identify if the author is 'Chris' or 'Sara' using SVM

Objective:

The objective of this exercise is to apply SVM to the email_text data as before and do the following:

1. Compare the accuracy with Gaussian NB that we used earlier
2. Speed up the algorithm
3. Tune up the Gamma, C parameters of the SVM for better performance

Data used:

Email_text data

Language:

python

Analysis:

Here we use an SVM identify emails from the Enron corpus by their authors:
(Sara has label 0 and Chris has label 1)

```
from sklearn.svm import SVC
clf = SVC(kernel='linear', C=10000.0)
#Train
clf.fit(features_train,labels_train)
#predict on Test data
pred = clf.predict(features_test)
#Calculate Accuracy
from sklearn.metrics import accuracy_score
acc = accuracy_score(labels_test, pred)
print "Accuracy for SVM:", acc
```

Results for above:

```
Kavyas-MacBook-Air:svm kavyagautam$ python svm_author_id.py
no. of Chris training emails: 7936
no. of Sara training emails: 7884
Training time for SVM: 199.386 s
Predicting time for SVM: 20.879 s
Accuracy for SVM: 0.984072810011
Kavyas-MacBook-Air:svm kavyagautam$
```

So we get a pretty good accuracy of 0.98, but the model took long enough to train on the training data.
So, we can speed up the algorithm by reducing the size of the data set (considering only 1% of previous training data). This will reduce the accuracy, though.

```
Kavyas-MacBook-Air:svm kavyagautam$ python svm_author_id.py
no. of Chris training emails: 7936
no. of Sara training emails: 7884
Training time for SVM: 0.117 s
Predicting time for SVM: 1.183 s
Accuracy for SVM: 0.884527872582
Kavyas-MacBook-Air:svm kavyagautam$
```

Now, lets change the Kernel to 'rbf' instead of 'linear'

```
Kavyas-MacBook-Air:svm kavyagautam$ python svm_author_id.py
no. of Chris training emails: 7936
no. of Sara training emails: 7884
Training time for SVM: 0.126 s
Predicting time for SVM: 1.344 s
Accuracy for SVM: 0.616040955631
Kavyas-MacBook-Air:svm kavyagautam$
```

That's a pretty drastic decrease in accuracy. So may be linear Kernel was more suitable

But, the C parameter gives different results for different values. So when tried with an increased C value of 10000, inclining towards more flexibility of model using rbf kernel, we get the following optimized results

C=10000, rbf kernel – Optimized SVM:

```
Kavyas-MacBook-Air:svm kavyagautam$ python svm_author_id.py
no. of Chris training emails: 7936
no. of Sara training emails: 7884
Training time for SVM: 132.13 s
Predicting time for SVM: 15.676 s
Accuracy for SVM: 0.990898748578
Kavyas-MacBook-Air:svm kavyagautam$
```

----------End of Section 2----------

Section 3. Decision Trees

3.1 Learning outcomes

The following topics were studied and practiced using some practice exercises and applied to the mini projects and the final Enron project

- Decision Tree – splits
- Data Impurity and Entropy
- Entropy calculation

Entropy = $\sum - P_i \log_2 (P_i)$

- Information gain calculation

Information gain = entropy of parent – [Weighted average] * entropy of children

- Tuning criterion for Decision tree classifier (gini index)
- Bias–Variance trade off

3.2 Practice mini project

3.2.1 Identify if the author is 'Chris' or 'Sara' using Decision Tree classifier

Objective:

In this mini project, we will again try to identify the authors in a body of emails, this time using a decision tree.

Data used:
Email_text data

Language:
python

Analysis:
The final code for decision tree classifier is as below and this has been arrived at after a series of steps of tuning and optimizing as follows:
1. Get a decision tree up and running as a classifier – This takes a while to train and predict
2. We know that parameter tuning can significantly speed up the training time of a machine-learning algorithm. There are 3785 features originally; removing some features can speed up the process.
selector = SelectPercentile(f_classif, percentile=10)
Changing the percentile from 10 to 1 can give us just 379 features.

```
from sklearn import tree
clf = tree.DecisionTreeClassifier(min_samples_split=40)
clf.fit(features_train, labels_train)

pred = clf.predict(features_test)

from sklearn.metrics import accuracy_score
acc = accuracy_score(labels_test, pred)

print "Accuracy of DT:", acc
  print len(features_train[0])
```

Results

```
Kavyas-MacBook-Air:svm kavyagautam$ cd ../decision_tree
Kavyas-MacBook-Air:decision_tree kavyagautam$ python dt_author_id.py

no. of Chris training emails: 7936
no. of Sara training emails: 7884
Accuracy of DT: 0.976678043231
Kavyas-MacBook-Air:decision_tree kavyagautam$
Kavyas-MacBook-Air:decision_tree kavyagautam$ python dt_author_id.py
no. of Chris training emails: 7936
no. of Sara training emails: 7884
3785
Kavyas-MacBook-Air:decision_tree kavyagautam$ python dt_author_id.py
no. of Chris training emails: 7936
no. of Sara training emails: 7884
379
Kavyas-MacBook-Air:decision_tree kavyagautam$ python dt_author_id.py
no. of Chris training emails: 7936
no. of Sara training emails: 7884
379
Accuracy of DT: 0.967007963595
Kavyas-MacBook-Air:decision_tree kavyagautam$
```

3. The final accuracy calculated after reducing the features is 96.7%, which worked well for this data compared to Support vector machines

----------*End of Section 3*----------

Section 4. Other ML Algorithms, Regression and more topics

4.1 Learning outcomes

Besides the three algorithms discussed in the above sections, the learning outcomes of this project also include other machine learning algorithms like:

- Adaboost – Ensemble trees
- Random forests – Ensemble trees
- K Nearest Neighbor classifier
- Text Learning (TFIDF vectorization)

Regressions:

- Continuous VS Discrete
- Slope and Intercept of regression line
- Predictions using regression
- Minimizing sum of squared errors
- R-Squared metric for regression
- Multivariate regression

Unsupervised Learning:

- K Means clustering

Some more important topics:

- Dealing with outliers
- Outlier detection and removal
- Slope and score of Regression with outliers VS without regression – coding
- Feature selection using Lasso regression
- Feature scaling
- Principal Component Analysis
- Validation and Cross-Validation
- Evaluation Metrics (Grid-Search)

All the above topics were studied and practiced using some practice exercises and applied to the mini projects and the final Enron project as applicable

4.2 Practice mini projects

In this part, we will see some of the practice work and comparison analysis done for some of the selected topics from the above list, for Enron Financial dataset

4.2.1 Visualizing Regression data – Slope and Regression Test Score

Objective:

In this mini-project 1, we will use the financial features of the Enron dataset and plot some regression lines and compare scores

Data used:

Enron Financial dataset

Language:

python

Analysis:

Step 1: Set up the features, Fit the regression

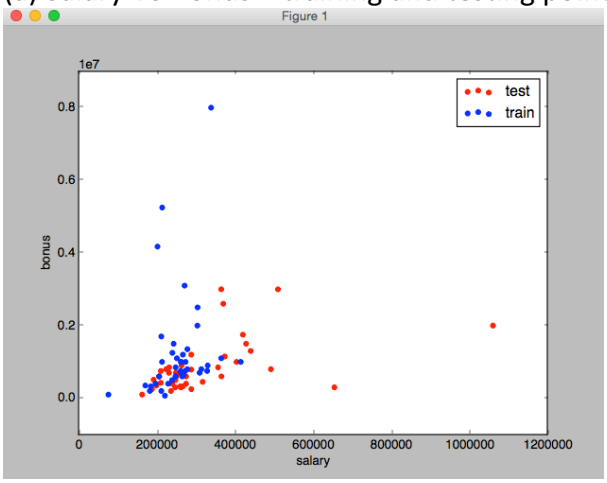
```
features_list = ["bonus", "salary"]
data = featureFormat( dictionary, features_list, remove_any_zeroes=True)
target, features = targetFeatureSplit( data )
```

Fitting the linear_model

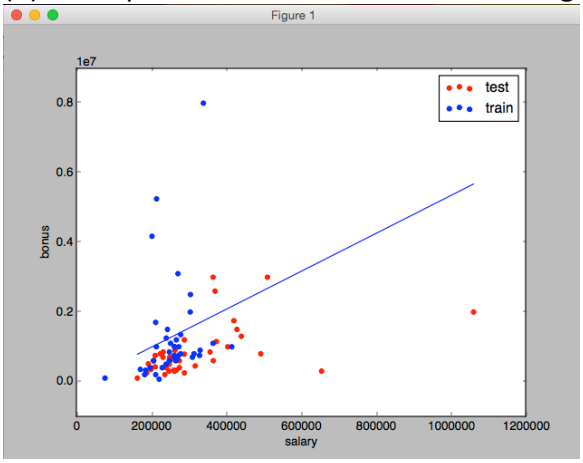
```
from sklearn import linear_model
reg = linear_model.LinearRegression()
reg.fit(feature_train, target_train)

print "Slope of the line:", reg.coef_
print "Intercept of the line:", reg.intercept_

print "Score on training data:", reg.score(feature_train, target_train)
print "Score on test data:", reg.score(feature_test, target_test)
```

Step 2: Plotting scatterplots of data and finding the Regression score

(a) Salary VS Bonus – training and testing points

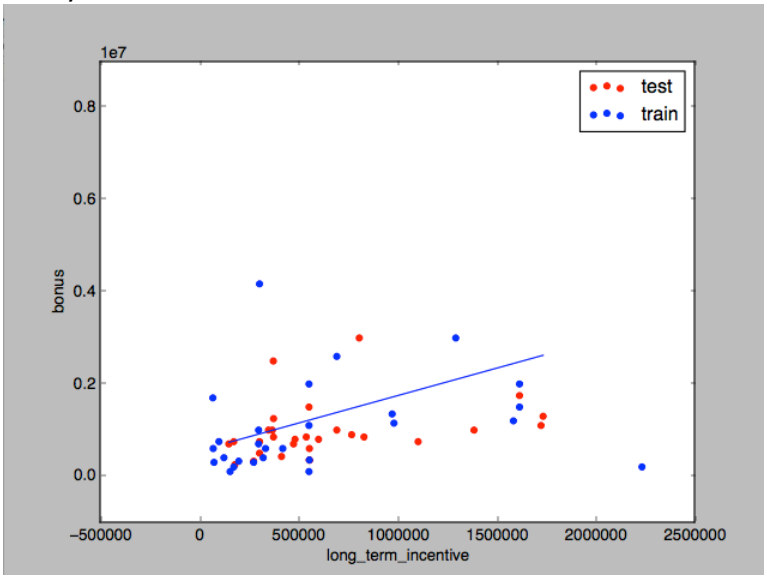


(b) Same points as above with the linear regression line fit



Slope of the line: [5.44814029]
Intercept of the line: -102360.543294
Score on training data:
0.0455091926995
Score on test data: -1.48499241737

(c) The same analysis as above, but now with Bonus VS long_term_incentive instead of salary



Slope of the line: [1.19214699]
Intercept of the line: 554478.756215
Score on training data: 0.217085971258
Score on test data: -0.59271289995

Step 3: Conclusion
The score for (c) is better than (b), So based on the above results, 'long_term_incentive' would a better feature for predicting someone's bonus rather than 'salary'

4.2.2 Visualizing outliers – Detection and removal

Objective:

In this mini-project 2, we will use the same Enron financial data and visually compare plots

Data used:
Enron Financial dataset

Language:
python

Analysis:
Step 1: Outlier detection in the Enron data

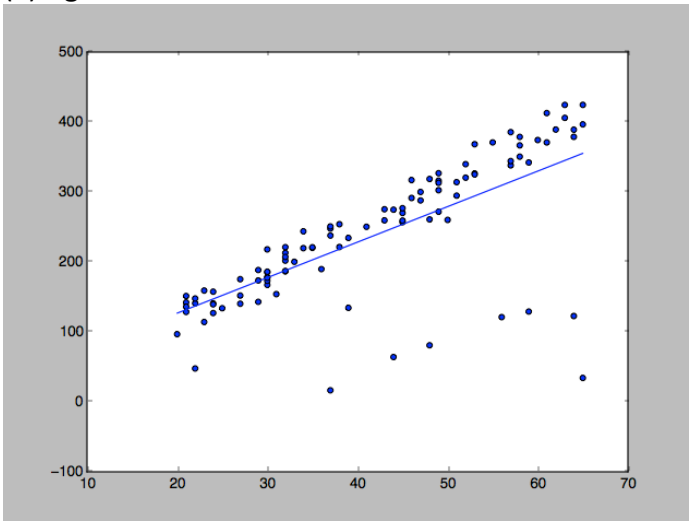
```
###We need to use data_dict and store salary outliers or bonus outliers into a list

outliers = []
for key in data_dict:
    value = data_dict[key]['salary']
    if value == 'NaN':
        continue
    outliers.append((key, value))
```

```
outliers = sorted(outliers, reverse = True, key = lambda tup:tup[1])

print "The name of the outlier point wrt salary:", outliers[:2]
```

```
outliers = []
for key in data_dict:
    value = data_dict[key]['bonus']
    if value == 'NaN':
        continue
    outliers.append((key, value))


outliers = sorted(outliers, reverse = True, key = lambda tup:tup[1])

print "The name of the outlier point wrt bonus:", outliers[:2]
```

Step 2: Visualization of outliers using scatterplots

(a) Ages VS Networths from enron data – with outliers



Slope of the regression line: [[5.07793064]]

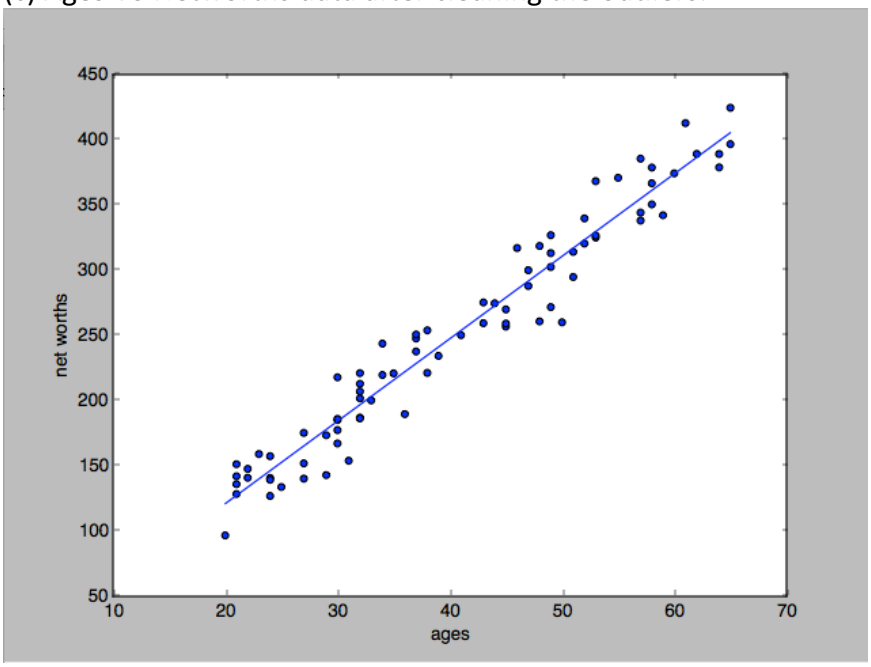
(b) Cleaning the outliers – using outlier removal regression algorithm:
Fit a regression

```
from sklearn import linear_model
reg = linear_model.LinearRegression()
reg.fit(ages_train, net_worths_train)
print "Score on test data:", reg.score(ages_test, net_worths_test)
print "Slope of the regression line:", reg.coef_


### identify and remove the most outlier-y points
cleaned_data = []
try:
    predictions = reg.predict(ages_train)
    cleaned_data = outlierCleaner( predictions, ages_train, net_worths_train )
except NameError:
    print "your regression object doesn't exist, or isn't name reg"
    print "can't make predictions to use in identifying outliers"
```

Clean away the 10% of points that have the largest residual errors (difference between the prediction and the actual net worth).

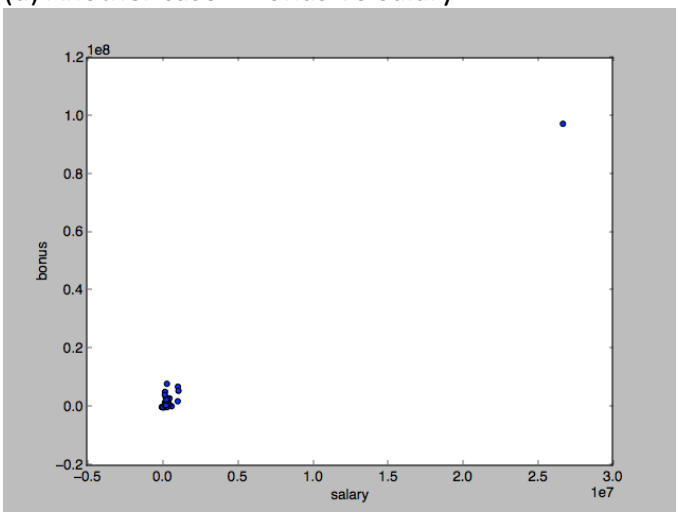
```
cleaned_data = []

### Outlier removal
residual_errors = list((net_worths - predictions)**2)
cleaned_data = zip(ages, net_worths, residual_errors)
cleaned_data = sorted(cleaned_data, key = lambda tup: tup[2])
cleaned_data = cleaned_data[:80]
```

(c) Ages VS Networths data after cleaning the outliers:

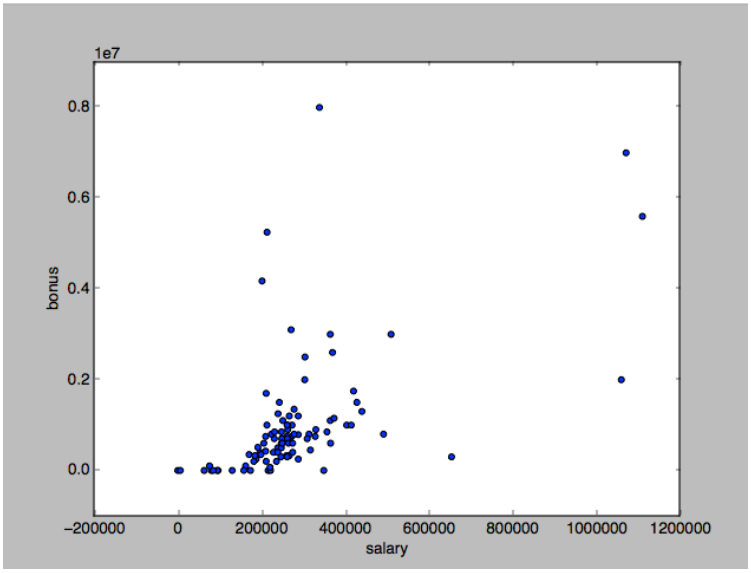


NEW Slope of the refitted regression line: [[6.32006691]]

(d) Another case – Bonus VS Salary



(e) Finding out the two main people who have huge bonuses and salaries



Kavyas-MacBook-Air:outliers kavyagautam$ python enron_outliers.py
The name of the outlier point wrt salary: [('SKILLING JEFFREY K', 1111258), ('LAY KENNETH L', 1072321)]
The name of the outlier point wrt bonus: [('LAVORATO JOHN J', 8000000), ('LAY KENNETH L', 7000000)]
SKILLING JEFFREY K and LAY KENNETH L
Their BONUS and SALARY
1072321 7000000
1111258 5600000

Step 3: Conclusion
We can now similarly identify all the outliers and remove all of those data points that have very high residual error values indicating that they are outliers.

----------End of Section 4----------

Section 5. Final Enron-Fraud Detection project

Identify Enron Employees who may have committed fraud based on the public Enron financial and email dataset.

5.1 Project Overview

In 2000, Enron was one of the largest companies in the United States. By 2002, it had collapsed into bankruptcy due to widespread corporate fraud. In the resulting Federal investigation, a significant amount of typically confidential information entered into the public record, including tens of thousands of emails and detailed financial data for top executives

In this project, all the learning outcomes above will be put together in building a person of interest identifier based on financial and email data made public as a result of the Enron scandal. We have available to us, a hand-generated list of persons of interest

in the fraud case, which means individuals who were indicted, reached a settlement or plea deal with the government, or testified in exchange for prosecution immunity.

The objective is to test various algorithms on the data and find one most suitable model that predicts the "Persons of Interest" most effectively (using evaluation metrics like "Accuracy", "Precision" and "Recall")

Data used:

- Enron Financial and email dataset in the form of pickle files (.pkl)
- Large text files, each of which contains all the messages to or from a particular email address

Features of data:

Combine the Enron email and financial data into a dictionary, where each key-value pair in the dictionary corresponds to one person. The dictionary key is the person's name, and the value is another dictionary, which contains the names of all the features and their values for that person. The features in the data fall into three major types, namely financial features, email features and POI labels.

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'] (all units are in US dollars)

Email features: ['to_messages', 'email_address', 'from_poi_to_this_person', 'from_messages', 'from_this_person_to_poi', 'poi', 'shared_receipt_with_poi'] (units are generally number of emails messages; notable exception is 'email_address', which is a text string)

Person Of Interest label: ['poi'] (boolean, represented as integer)

5.2 Solution - Breakdown

As seen, in the practice exercises and mini projects earlier, the steps involved in arriving at the resulting algorithm are fairly the same.

STEP 1: SELECT FEATURES, REMOVE OUTLIERS AND ADD ADDITIONAL FEATURES

Various techniques can be applied for feature selection

- Intuition and visual inspection
- Lasso's Regression and K best feature selection
- Mix of both the steps above

```
### The first feature must be "poi".
features_list = ['poi'] # This is the target label.

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

1. Outlier detection and removal for the actual Enron dataset is done as follows:

```
num_data_points = len(data_dict)
num_data_features = len(data_dict[data_dict.keys()[0]])

num_poi = 0
for dic in data_dict.values():
    if dic['poi'] == 1: num_poi += 1

print "Data points: ", num_data_points
print "Features: ", num_data_features
print "POIs: ", num_poi
```

```
employee_names = []
for employee in data_dict:
    employee_names.append(employee)
employee_set = set(employee_names)

#Sort alphabetically and visually inspect
employee_names.sort()
#print(employee_names)

###Remove the outlier "TOTAL" from the original dataset
data_dict.pop('TOTAL', 0)
print "Data points after removing outlier", len(data_dict) #144 records as expected
```

2. Now we can create two new additional intuitive features to the existing feature_list

```
my_dataset = data_dict

## ADDITIONAL FEATURE 1
## Add the fraction of emails from poi and to poi as an email feature
def computeFraction( poi_messages, all_messages ):
    """ given a number messages to/from POI (numerator)
        and number of all messages to/from a person (denominator),
        return the fraction of messages to/from that person
        that are from/to a POI
   """
    ### beware of "NaN" when there is no known email address (and so
    ### no filled email features), and integer division!
    ### in case of poi_messages or all_messages having "NaN" value, return 0.
    fraction = 0

    if (poi_messages == 'NaN' or all_messages == 'NaN'):
        return 0
    else:
        fraction = poi_messages*1.0/all_messages

    return fraction
```

```
###Actual creation of feature happens here
for name in my_dataset:

    data_point = my_dataset[name]

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


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

```
## ADDITIONAL FEATURE 2
"""
And as a financial feature, we can add the total assets value which intuitively
will be the sum of 'salary', 'bonus', 'total_stock_value' """

for name in my_dataset:
    data_point = my_dataset[name]
    if (all([   data_point['salary'] != 'NaN',
                data_point['exercised_stock_options'] != 'NaN',
                data_point['total_stock_value'] != 'NaN',
                data_point['bonus'] != 'NaN'
            ])):
        data_point['wealth'] = sum([data_point[each] for each in ['salary',
                                                                  'exercised_stock_options',
                                                                  'total_stock_value',
                                                                  'bonus']])
    else:
        data_point['wealth'] = 'NaN'
```

So, with this, we add two new intuitive features to the existing feature_list

3. Now, its time to add these two new features along with the existing features to our dataset:

```
### Now add these above features + some more additional features to the feature_list
features1 = features_list + ['fraction_from_poi',
                             'fraction_to_poi',
                             'shared_receipt_with_poi',
                             'expenses',
                             'loan_advances',
                             'long_term_incentive',
                             'restricted_stock',
                             'salary',
                             'total_stock_value',
                             'exercised_stock_options',
                             'total_payments',
                             'bonus',
                             'wealth']

### Extract features and labels from dataset for local testing
data = featureFormat(my_dataset, features1, sort_keys = True)
labels, features = targetFeatureSplit(data)
```

We do not know yet if feature scaling and feature filtering using kbest will benefit our model yet. So lets try it anyway.

```
# Scale features
scaler = MinMaxScaler()
features = scaler.fit_transform(features)

# K-best features - choosing 6 features for a trial
k_best = SelectKBest(k=6)
k_best.fit(features, labels)

result_list = zip(k_best.get_support(), features1[1:], k_best.scores_)
result_list = sorted(result_list, key=lambda x: x[2], reverse=True)
#print "K-best features - i.e. top 6 features selected:", result_list

"""
OUTPUT:
K-best features - i.e. top 6 features selected:
[(True, 'exercised_stock_options', 25.097541528735491),
(True, 'total_stock_value', 24.467654047526391),
(True, 'bonus', 21.060001707536578),
(True, 'wealth', 19.457343207083316),
(True, 'salary', 18.575703268041778),
(True, 'fraction_to_poi', 16.641707070468989),
(False, 'long_term_incentive', 10.072454529369448),
(False, 'restricted_stock', 9.3467007910514379),
(False, 'total_payments', 8.8667215371077805),
(False, 'shared_receipt_with_poi', 8.7464855321290802),
(False, 'loan_advances', 7.2427303965360172),
(False, 'expenses', 6.234201140506757),
(False, 'fraction_from_poi', 3.2107619169667667)]
"""

## 6 best features chosen by SelectKBest
features2 = features_list + ['exercised_stock_options',
                             'total_stock_value',
                             'bonus',
                             'wealth',
                             'salary',
                             'fraction_to_poi']
```

As it turns out, none of the algorithms that will be shown below benefit from feature scaling and selecting 6 best for the reasons that the number of persons of interest is actually quite low compared to the actual employee count in the data.
Nor does the inclusion of entire list of features helps.

So, as a final attempt at feature selection, I settle at 4 intuitive features and add them to the data set as follows:

```
## Finally my features that sound intuitive to me
my_features = features_list + ['wealth',
                               'fraction_to_poi',
                               'fraction_from_poi',
                               'shared_receipt_with_poi']
print ""
print "Selected Feature list - By Intuition", my_features

### Extract features and labels from dataset for local testing
data = featureFormat(my_dataset, my_features, sort_keys = True)
labels, features = targetFeatureSplit(data)
```

STEP 2: TRY A VARIETY OF CLASSIFIERS

Here, we will try some of the classifiers learnt and using some evaluation techniques, analyze the performance of each classifier
Splitting up the data into training and testing data sets:

```
from sklearn.cross_validation import train_test_split
features_train, features_test, labels_train, labels_test = \
    train_test_split(features, labels, test_size=0.3, random_state=42)

print "**************************"
test_classifier(clf, my_dataset, my_features)
```

1. NAIVE BAYES

```
"""
Classifier Number 1 - Naive Bayes
"""
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()
parameters = {}
grid_search = GridSearchCV(clf, parameters)
print '\nGaussianNB:'
test_clf(grid_search, features, labels, parameters)
```

```
#*****************************************************************************************************************
"""
Classifier Number 1 - Naive Bayes
"""
# OUTPUT:
"""
GaussianNB()
    Accuracy: 0.84364   Precision: 0.42879  Recall: 0.28450 F1: 0.34205 F2: 0.30503
    Total predictions: 14000    True positives:  569    False positives:  758   False negatives: 1431
    True negatives: 11242
"""
```

2. DECISION TREES

```
"""
***This is the final Chosen Classifier becaue of a sizeable and comparable Precision and Recall
Classifier Number 2 - Decision Tree
"""
from sklearn import tree
clf = tree.DecisionTreeClassifier()

parameters = {'criterion': ['gini', 'entropy'],
              'min_samples_split': [2, 10, 20],
              'max_depth': [None, 2, 5, 10],
              'min_samples_leaf': [1, 5, 10],
              'max_leaf_nodes': [None, 5, 10, 20]}
grid_search = GridSearchCV(clf, parameters)
print '\nDecisionTree:'
test_clf(grid_search, features, labels, parameters)
```

```
#*****************************************************************************************************
#***This is the final Chosen Classifier because of a sizeable and comparable Precision and Recall values
"""
Classifier Number 2 - Decision Tree - WINNER
"""
# OUTPUT:
"""
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=None, splitter='best')
    Accuracy: 0.88180   Precision: 0.40733  Recall: 0.40000 F1: 0.40363 F2: 0.40145
    Total predictions: 10000    True positives:  400    False positives:  582   False negatives:  600
    True negatives: 8418
"""
```

3. ADABOOST

```
"""
Classifier Number 3 - AdaBoost Classifier - Ensembles of trees
"""
from sklearn.ensemble import AdaBoostClassifier
clf = AdaBoostClassifier()
parameters = {'n_estimators': [10, 20, 30, 40, 50],
              'algorithm': ['SAMME', 'SAMME.R'],
              'learning_rate': [.5,.8, 1, 1.2, 1.5]}
grid_search = GridSearchCV(clf, parameters)
print '\nAdaBoost:'
test_clf(grid_search, features, labels, parameters)
```

```
#*****************************************************************************************************
"""
Classifier Number 3 - AdaBoost Classifier - Ensembles of trees
"""
# OUTPUT:
"""
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
          learning_rate=1.0, n_estimators=50, random_state=None)
    Accuracy: 0.82550   Precision: 0.34521  Recall: 0.24700 F1: 0.28796 F2: 0.26190
    Total predictions: 14000    True positives:  494    False positives:  937   False negatives: 1506
    True negatives: 11063
"""
```

4. K-NEAREST NEIGHBOR

```
"""
Classifier Number 4 - K nearest Neighbor Classifier
"""
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier()
parameters = {'algorithm' : ['auto', 'kd_tree'],
              'leaf_size' : [30, 40],
              'metric' : ['minkowski'],
              'n_neighbors' : [2, 3, 4, 5, 6, 7],
              'p' : [2],
              'weights' : ['uniform','distance']}
grid_search = GridSearchCV(clf, parameters)
print '\nKNeighborsClassifier:'
test_clf(grid_search, features, labels, parameters)
```

```
#*********************************************************************************************************
"""
Classifier Number 4 - K nearest Neighbor Classifier
"""
# OUTPUT:
"""
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')
    Accuracy: 0.85264   Precision: 0.38628  Recall: 0.05350 F1: 0.09398 F2: 0.06464
    Total predictions: 14000    True positives:  107    False positives:  170   False negatives: 1893
    True negatives: 11830
"""
```

5. SUPPORT VECTOR MACHINES

```
"""
Classifier Number 5 - Principal Component Analysis and Support Vector Machine
"""
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
estimators = [('reduce_dim', PCA()), ('svm', SVC())]
clf = Pipeline(estimators)
parameters = dict(reduce_dim__n_components=[2, 5], svm__C=[0.1, 10, 100])
grid_search = GridSearchCV(clf, parameters)
print '\nPCA and SupportVectorMachine:'
test_clf(grid_search, features, labels, parameters)
```

```
#*********************************************************************************************************
"""
Classifier Number 5 - PCA & Support Vector machines
"""
# OUTPUT:
"""
Got a divide by zero when trying out: Pipeline(steps=[('reduce_dim', PCA(copy=True, n_components=None,
  decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False))])
Precision or recall may be undefined due to a lack of true positive predicitons.
"""
.........
Precision: 0.0
Recall: 0.0
reduce_dim__n_components=2,
svm__C=0.1,
"""
```

STEP 3: CONCLUSION

After analyzing the true positives, false positives from the above metrics of each algorithm, we can understand that, Precision and Recall matter more than overall Accuracy when identifying a person as potential fraud based on his/her email/financial data. (Even though the accuracy is 88%, we have to look at best precision and recall)

With respect to Precision and Recall, ***Classifier 2 - Decision Tree*** clearly stands as the winner with both precision and recall values of 0.4, which means, 40% of who our model identifies as POIs turned out to be really the ones who were indicted in real-time.

----------End of Section 5----------