INTRODUCTION TO MACHINE LEARNING, FINAL PROJECT

STUDENT: Guillermo Naranjo

DATE: Sunday, Nov 12th, 2017.

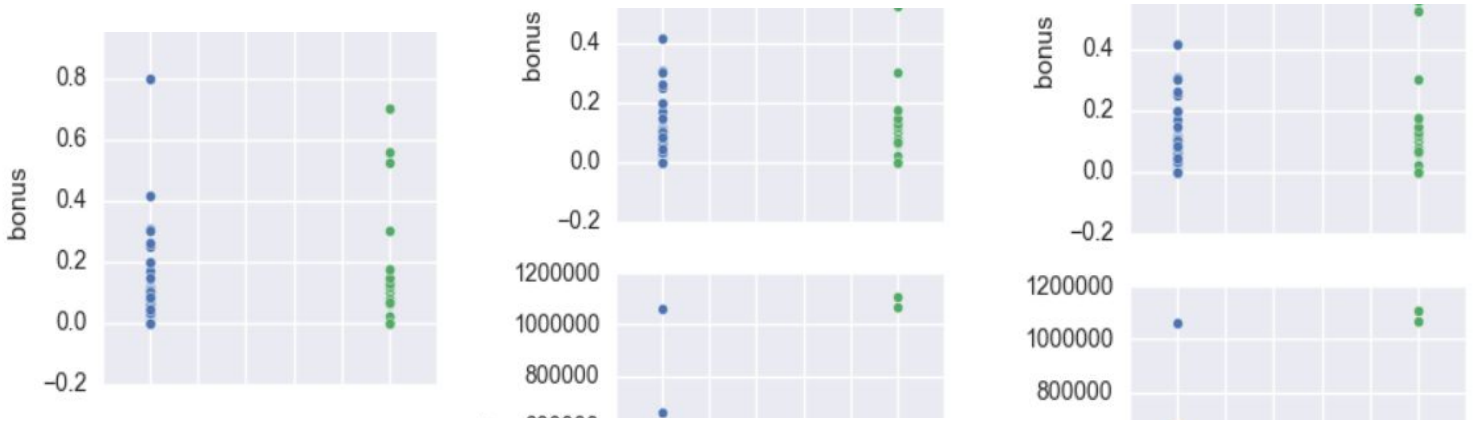
Question #1 Summarize for us the **goal** of this project and how machine learning is useful in trying to accomplish it. As part of your answer, give some background on the dataset and how it can be used to answer the project question. Were there any **outliers** in the data when you got it, and how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

Goal: The main objective of this project is building a person of interest identifier based on financial and email data made public as a result of the Enron scandal. It was useful to apply machine learning since we have 21 attributes either email or financial, quite complex to analyze manually. It provides an iterative and efficient way to predict a POI in this case.

NaN: Additionally, attributes like director_fees were discarded because they had several NaN, with 0 or 18 related POIs. If included they could create unbalanced results in the model.

	feature	total	wo_nan	w_nan	poi	wo_nan_poi	w_nan_poi
0	bonus	146	81	64	18	16	2
1	deferral_payments	146	38	107	18	5	13
2	deferred_income	146	48	97	18	11	7
3	director_fees	146	16	129	18	0	18
4	exercised_stock_options	146	101	44	18	12	6
5	expenses	146	94	51	18	18	0
6	from_messages	146	86	59	18	14	4
7	from_poi_to_this_person	146	74	71	18	14	4
8	from_this_person_to_poi	146	66	79	18	14	4
9	loan_advances	146	3	142	18	1	17
10	long_term_incentive	146	65	80	18	12	6
11	other	146	92	53	18	18	0
12	poi	146	18	127	18	18	0
13	restricted_stock	146	109	36	18	17	1
14	restricted_stock_deferred	146	17	128	18	0	18
15	salary	146	94	51	18	17	1
16	shared_receipt_with_poi	146	86	59	18	14	4
17	to_messages	146	86	59	18	14	4
18	total_payments	146	124	21	18	18	0
19	total_stock_value	146	125	20	18	18	0

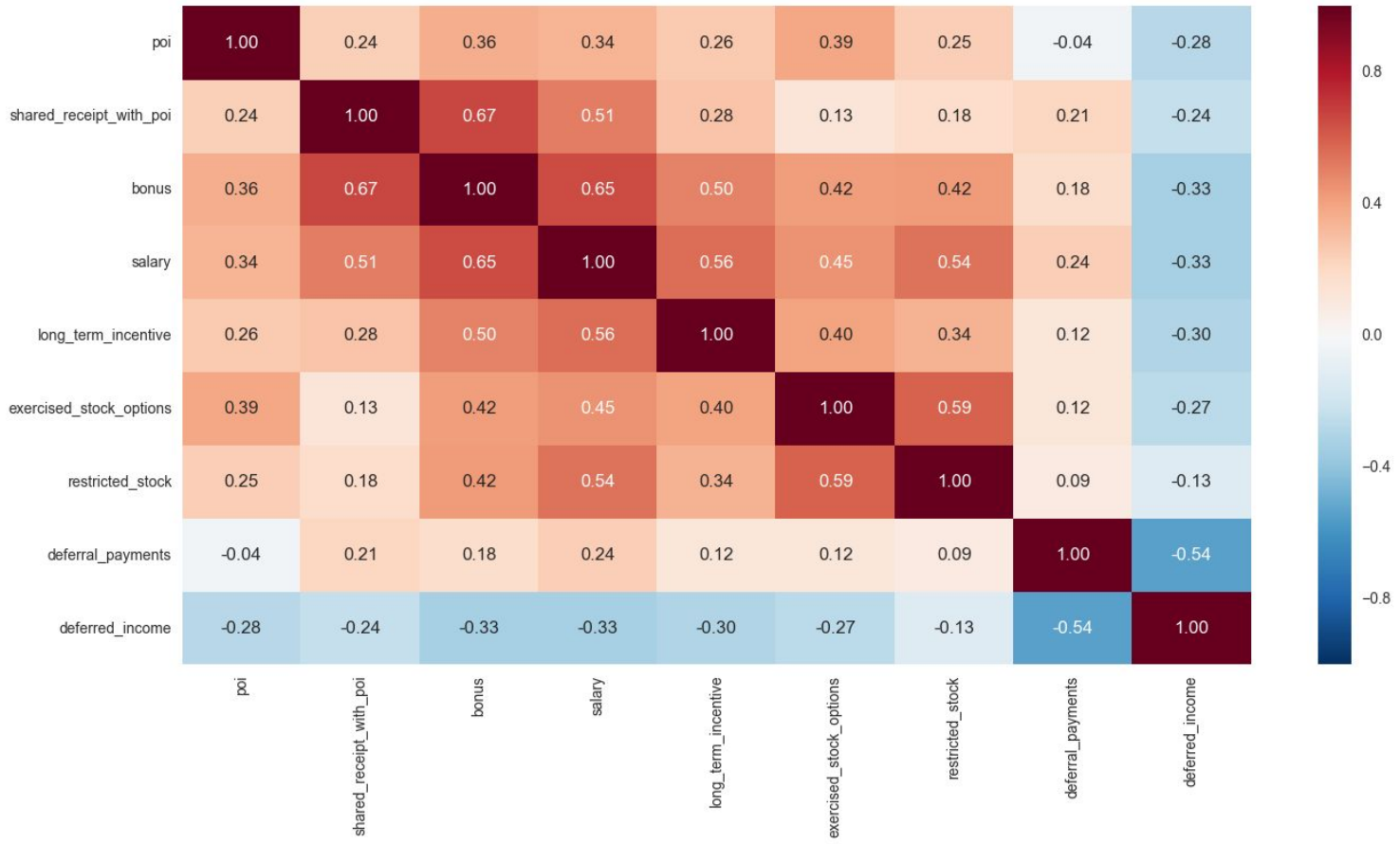
Outliers: As you can see in “exploration.html” file, the one outlier removed was TOTAL as explained in the course. Features like salary, bonuses and long_term_incentive showed up to three possible outliers but once verified they seems to be valid and possible POIs, so, I'll keep them in. Once removed TOTAL outlier there are 146 remaining observations.



That been said, only bonus, deferral_payments, deferred_income, exercised_stock_options, from_messages, rom_poi_to_this_person, from_this_person_to_poi, long_term_incentive, restricted_stock, salary, to_messages, and shared_receipt_with_poi remained after initial manual univariate analysis.

Question #2. What features did you end up using in your POI identifier, 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”]

Initial feature selection: after initial analysis, these features were evaluated to identify noise, high correlation and other patterns, :



Decisions:

- I decided to use deferred_income and not deferral_payments since it has more balanced POIs/NO POIs.
- There is a relation between bonus, salary so I decided to engineer a new feature (additional salaries per bonus) since the is a strong relation between them.
- Shared_receipt_with_poi is a good representation of email data and has good relation to POI.
- Exercise_stock_options by itself is the one with the highest relation to POI it need to be included.
- I tried to define at least 4 or 5 independent variables in order to test GaussianNB as well since it's a requirement for that model and see how it performs with other models.

Then I used both SelectKBest (k=5) and SelectPercentile (p=50) to validate my selection, resulting, I sed f_classif:

ORIGINAL FEATURES: ['bonus_extra_salary', 'shared_receipt_with_poi', 'deferred_income', 'exercised_stock_options', 'long_term_incentive', 'bonus', 'salary', 'deferral_payments']
INTELLIGENT FEATURE SELECTION:

- SelectPercentile:(percentaje=50) ['deferred_income', 'exercised_stock_options', 'bonus', 'salary']
- SelectKBest(k=5): **['bonus_extra_salary', 'deferred_income', 'exercised_stock_options', 'bonus', 'salary']**

Question #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"]

Initial results: I trained and tested GaussianNB, DecisionTreeClassifier and SVC classifiers. After initial exploration I decided to go with GaussianNB:

- Simple calls to the methods GaussianNB, DecisionTree and SVC RBF with initial exploration features.
- No SelectKBest suggested features. No hyperparameter tuning. Simple train_test_split.
- Features used: ['bonus_extra_salary', 'shared_receipt_with_poi', 'exercised_stock_options', 'long_term_incentive']

GaussianNB:	Accuracy	Precision	Recall	F1
Basic Version	0.871794871795	0.333333333333	0.25	0.285714285714

- Then I decided to tune the algorithm by using PCA randomized with 2 PC. I mainly used PCA to increase independence between features even more. Results improved to:

GaussianNB:	Accuracy	Precision	Recall	F1
PCA Version	0.880952380952	0.555555555556	0.833333333333	0.666666666667
	Explained variance ratio pc: [0.97661654 0.02338329] First pc: [1.42239590e-07 3.07584610e-05 -1.40070448e-01 9.90141540e-01] Second pc: [2.64910114e-06 7.86353015e-04 -9.90141231e-01 -1.40070429e-01]			

- Used suggested features in SelectBestK and GridSearchSV(average_precision) and test.py returned.

GaussianNB:	Accuracy	Precision	Recall	F1
SelectKBest(5)+ PCA(5) + StratifiedKFold + GridSearch (**precision**) Version	0.86743	**0.55607**	0.35700	0.43484
	The best parameters are {'reduce_dim__n_components': **5**} with a score of **0.66** ['bonus_extra_salary', 'deferred_income', 'exercised_stock_options', 'bonus', 'salary']			

I also tested **other classifiers**:

DecisionTreeClassifier:	Accuracy	Precision	Recall	F1
Basic no tuning	0.820512820513	0.2	0.25	0.222222222222
	Importances: bonus_extra_salary : 0.178646793292 shared_receipt_with_poi : 0.333557602277 exercised_stock_options : 0.293300954344 long_term_incentive : 0.194494650087			

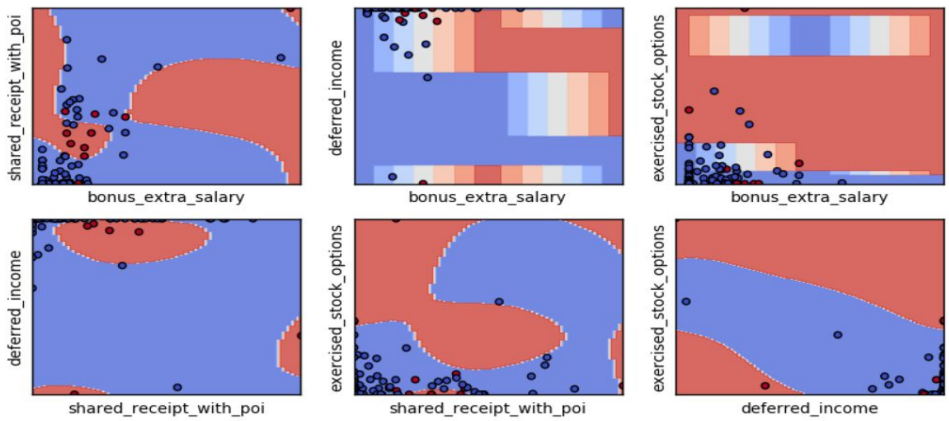
- In this case I tried to tune the DTree sorting and using most important features and removing long_term_incentive. I included deferred_income while validating importances with several features and results improved:

DecisionTreeClassifier:	Accuracy	Precision	Recall	F1
min_samples_split =2 s_depth=3	0.880952380952	0.571428571429	0.666666666667	0.615384615385
	Importances: bonus_extra_salary : 0.248690904897 shared_receipt_with_poi : 0.253691883522 deferred_income : 0.316778127837 exercised_stock_options : 0.180839083745			

- Initially I liked Decision Trees but the are very unstable from fold to fold, and difficult to tune, you need little noise, good defined features and quality data, things that we don't particularly have in ENRON dataset.

SVC:	Accuracy	Precision	Recall	F1
No Tuning	0.897435897436	0.0	0.0	0.0

- It was really poor so I decided to apply feature scaling and PCA first to normalize values to help RBF kernel and then PCA to reduce noise and complexity by using two components.
- I tested GridSearchCV with several values for C and Gamma without good results. At the end I manually defined parameters C 2450 and gamma 5.7 by looking at this plot:



- Results improved to:

SVC:	Accuracy	Precision	Recall	F1
C 2450 gamma 5.7 PCA n=2	0.809523809524	0.333333333333	0.333333333333	0.333333333333

- SVC is really interesting but slow and very difficult to optimize (at least for a beginner).
- I tried linear kernel but i took more than 30 minutes to compute.

Question #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? [relevant rubric items: "discuss parameter tuning", "tune the algorithm"]

Tuning the parameters are the actions to apply different combination of inputs to achieve the most desirable output (defined by a suitable metric) consistently once applied new unseen test data. You use samples of the dataset to train the algorithm by learning the features patterns iteratively. Once it fits then you apply this fit to a unseen test sample in order to predict the output class. Without tuning hyperparameters the models would not predict the output class correctly and either it would be very unresponsive to new test data (high bias) or too responsive to new testing data (high variance), there would not be balance.

Tuning GaussianNB:

1. Selecting the best features using SelectBestK
2. Using stratifiedShuffleSplit to balance error and deviations to get the nest variance and bias
3. I used PCA to increase features independence and to reduce noise from less important variables.
4. Applying gridSearchCV to select the right number of PCA components (5) to achieve the highest **average_precision** ratios.

GaussianNB allowed me to get the best results with a good mix of bias (responds to new test data) and variance (responds differently to new test data) while keeping desired output metrics ratios.

Question #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 are methods to assess the performance of the learning algorithm to verify if it does what we **want** to do once learning from trained data the patterns from input variables and its relations to predict an output variable. Let's say we **want** our algorithm to predict with more than 50% precision the right POIs given a Enron database, the algorithm must reach consistently that metric with new unseen test data.

So as you can see here, **split** in training and testing data is very important because our algorithm must learn from training data and predict from test data to validate performance.The more trained data the more it would learn patterns and changes, getting balanced and flexible enough to predict right once applied independent sets.

If not applied correctly it would learn rigid rules from a few trained data, overfitting and not returning good results from unseen data. One classic mistake is to use sorted data, if not **shuffled** it would lead to unbalanced in training and test sets resulting in abnormally low or high metrics. Splitting datasets, using StratifiedKFold training sets and shuffling data are good practices to increase balance in our outcome clasess.

Finally, another way to apply cross-validation is by tuning hyperparameters of the model (gamma or C in SVM for example) by finding the parameter combination that delivers the highest performance once tested (defining the desired **scoring** parameter). I used functions like gridSearchCV to iteratively find the right number of PCA components that delivered the highest Precision in my GaussianNB classifier model.

I used these techniques:

- StratifiedShuffleSplit to "assign data points to folds so that each fold has approximately the same number of data points of each output class".
- GridSearchCV to tune hyperparameters along with StratifiedShuffleSplit as cross-validation generator.
- I also used a train_test_split to split the dataset in 70% train data 30% test data. It was useful to train and test results fast once the model was tuned.
- One thing that I would like to do in the future is to apply fold manually, calculating the average error and deviation for each fold in order to plot the balance (bias vs variance).

Question #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"]

- After my first review I'll take **precision** and **recall**.
- Using the selected model I got in test.py:
    - Precision: 0.55607
        - It shows how correct is the model on predicting the correct POI from the predicted total of POIs observations in the tested sample. For example, if we have to predict POIs from Enron employees dataset, the system predicted that there were 100 POIs but in reality there are only 55 correct POIs.
    - Recall: 0.35700
        - Is shows how "complete" is the model is in selecting the maximum number of correct POIs from the total number of POIs available in the test sample. For example, from a test sample that contains 100 real Enron POIs, the model is predicting 35 of them as real POIs. It's missing 65 that are real POIs.

Since the objective here is to identify a POI correctly based on the data provided without impacting those people that are not related to the fraud I would say **Precision is an important** metrics because is desired if I miss some of the real POIs (recall) but **is not ok to say someone is a POIs when in reality is not a POI.**