Identifying Fraud From Enron Email

Udacity Nanodegree Project 4

Coursework

- Intro to Machine Learning

By: Joe Nyzio

Identify Fraud from Enron Email

Udacity Nanodegree

By Joe Nyzio

View Project IPython Notebook
http://nbviewer.ipython.org/gist/JoeNyzio/3d3b0499837461aa125d#

Answers to Project Questions

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"]

Summarize

After the collapse of Enron a large dataset containing a variety of information was made publicly available. The goal of this project was to find a way to spot a person of interest from this dataset. We were provided emails, salaries, bonuses, and other financial information about Enron employees. I'll use machine learning to create a classification system that can help determine if someone is a person of interest using a particular set of features from the given data.

Background

In the dataset we have 146 people, 21 features, and the emails of 18 out of 35 of the poi's. I used IPython/Pandas to take a further look into the structure so I could use the information for analysis.

Outliers

There were a few outliers in the dataset but I chose to keep most them since they were legitimate pieces of information that I could use when creating my poi identifier. I did find one outlier when investigating the 'salary' and 'bonus' columns which I removed. I found this after noticing an extreme outlier in a graph of these variables. To get rid of it I populated a dictionary with the keys from the variables 'salary' and 'bonus' in descending order and removed the 0th index. This was a key named 'total' which was a sum of the variables entire set of entries. I removed it because it would negatively affect the results of any poi identification algorithm I would build.

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 doesn't come ready made in the data set 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.) If you used an algorithm like a decision tree, please also give the feature importances of the features that you use. [relevant rubric items: "create new features", "properly scale features", "intelligently select feature"]

Features

In my list of features I chose to focus on communication as the primary poi predictor. The 3 features most relevant to communication in the dataset are 'from this person to poi', 'from poi to this person', and 'shared receipt with poi'. I tried a mixture of other features based on similar intuitions and kept the ones the produced the best results.

Selecting Features

Features were selected by hand using my own intuition and understanding of the dataset. I tested out different feature combinations to confirm my intuition and help improve the accuracy of the model. The final model is a result of the keeping the best result from this process.

Scaling

I did not use feature scaling. The 2 algorithms I attempted were GaussianNB and Decision Trees. Decision trees split data points either vertically or horizontally, but not both simultaneously. This means there is no gradient implemented during this split so the relationship between the features is not applicable. Because of this the decision tree is unaffected by feature scaling. The naive bayes algorithm follows similar principles that ignore the relationship between features. Features in the model are scaled automatically depending on their assigned coefficients which renders feature scaling useless.

Engineer your own feature

I tried to reduce the variables relating to emails sent by a worker to a single ratio of poi contacts/non poi contacts. I took the sum of to and from poi messages and divided it by the sum of the to and from non poi messages and saved it as poi_ratio.py. This feature improved the accuracy of my model and was used in the final algorithm.

3. What algorithm did you end up using? What other one(s) did you try? [relevant rubric item: "pick an algorithm"]

I ended up using a decision tree. I started of by using GaussianNB for its simplicity. Once I had a high Gaussian accuracy I fed the features I found most relevant into a decision tree so I could tune the parameters to get a higher precision and recall.

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 don't 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 if you used, say, a decision tree classifier). [relevant rubric item: "tune the algorithm"]

Tune the Parameters

Each algorithm is a function that we are calling with a few inputs that have been made available for us to tweak. They start off set to their default but once we begin changing these parameters around it's known as 'tuning the algorithm'.

Tuning the algorithm should be an attempt to optimize its performance so if you don't do this correctly you'll end up with a suboptimal algorithm and you should be ashamed of yourself.

My Algorithm

Decision trees have a variety of tunable parameters. I chose to play around with 'criterion' which can be either 'entropy', or 'gini', max_depth, which specifies how many levels the tree can go, and min_samples_leaf which is the minimum number of samples needed to split an internal code. I chose them because they were all above the fold on the sklearn documentation page. Thats probably not the best way to decide but I ended up with an acceptable algorithm with these parameters so I stuck with them.

I wrote down the accuracy, precision, and recall of all my tweaks and chose the best one. The final model I chose had the highest overall performance of all 3 parameters.

Accuracy: 0.77967 Precision: 0.50516 Recall: 0.41600

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

Validation is the process of figuring out if your algorithm is useful outside your specific dataset. Given a limited dataset, we need to partition it into a portion that we'll use to train the algorithm, and a portion that we'll use as a first run. This portion of the data is supposed to help us figure out how the algorithm would perform in the real world.

Classic Mistake

Doing this correctly gives us an estimate of performance on an independent dataset and serves as a check for overfitting. A classic mistake that can happen if we do this wrong is that our algorithm will overfit to the data it was created on and be unable to perform in real world situations.

Validate your analysis

I validated my analysis by comparing the accuracy, precision, and recall of the different clf's I had created. The algorithm I chose was the one most suitable to the project based on these criteria.

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"]

2 Evaluation Metrics

2 evaluation metrics are **precision** and **recall**.

Precision is the amount of true positives divided by the true positives + false positives.

Precision is like if you blindfolded a basketball player and asked them to say yes only if they thought they made their last shot. You'd end up with yeses that were right (true positives), yesses that were wrong(false positive), and probably an injured basketball player.

Now take that shooter and make him say yes or no to weather or not he made each shot. Now look at the times he made the shot and said he did (true positives), and the number of times he made a shot when he thought he didn't (false negatives).

You'd end up with a list of times he was right about making the shot, and a number of times he made a shot when he thought he didn't.

Now you can get the **recall** by taking those true positives and dividing them by the true positives plus the false negatives.

A shooter with low confidence in himself would probably say he missed a lot, have a bunch of false negatives, and is like an algorithm with low recall. Giving them a confidence boost might get their recall up unless they actually never make any shots.

Performance

My precision is .40 My recall is .349.