|  |
| --- |
| Final project on “introduction to Machine Learning” |
| Ahmed Alhuwaymil |

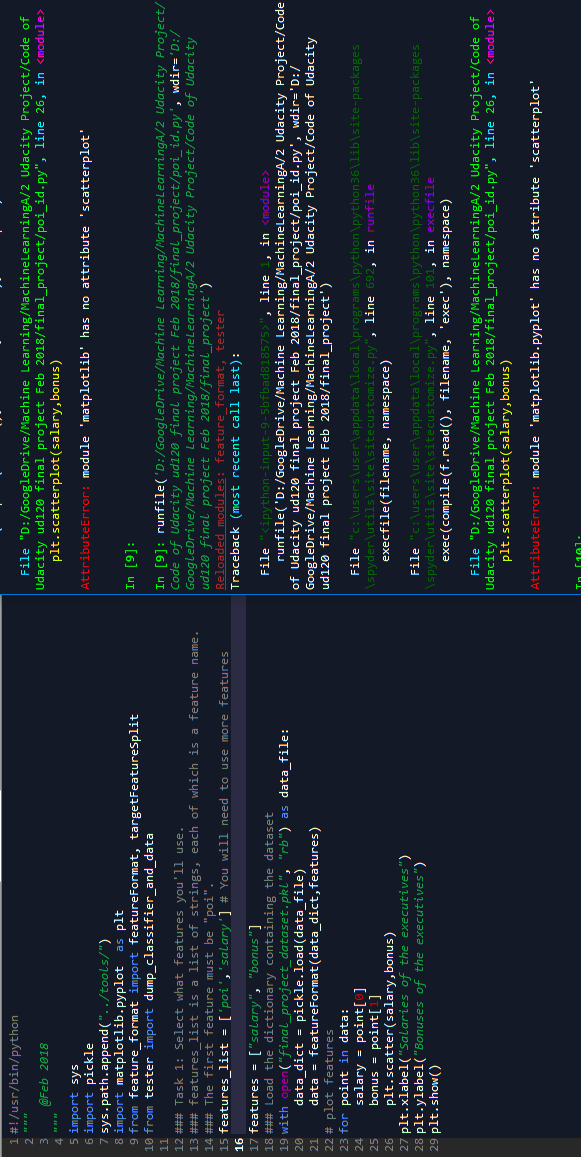
**

Table of Contents

[An Overview of the ENRON Project 3](#_Toc505628195)

[Exploring the data 3](#_Toc505628196)

[Detecting outliers and NaN values 3](#_Toc505628197)

[Feature Engineering 5](#_Toc505628198)

[Introducing new features 5](#_Toc505628199)

[Intelligently select features 6](#_Toc505628200)

[Scaling the feature 7](#_Toc505628201)

[Selecting final features 7](#_Toc505628202)

[Machine Learning Algorithms to implement 8](#_Toc505628203)

[Pick an algorithm 8](#_Toc505628204)

[Tune the algorithm 9](#_Toc505628205)

[Validation and Evaluation of the algorithm 10](#_Toc505628206)

[Discussion and Conclusion 10](#_Toc505628207)

# An Overview of the ENRON Project

Enron was the infamous American energy company that was accused to be the largest bankruptcy case in US history, totaling losses around 66 billion US dollars [[[1]](#endnote-1)], compelling 4,000 unemployed [[[2]](#endnote-2)]. Its collapse affected thousands of employees and shook Wall Street to its core. The primary goal of this Enron machine learning project is to extract meaningful information from ENRON dataset[[3]](#footnote-1). A machine learning model has been built which can predict probable person of interests from new dataset with satisfactory level of accuracies.

# Exploring the data

The Enron dataset contains a **total of 146 data points** (people as data point), each has 21 features. Both financial and email features are included in the dataset. It is assumed that there are 18 persons of interest (provided in the root directory [[[4]](#endnote-3)] of the project) out of total 146 persons or executives. The very high level overview of the dataset, obtained from [[[5]](#endnote-4)], is tabulated in Table 1: Overview of the dataset[[6]](#footnote-2). This dataset is useful to build a machine learning algorithm to predict possible criminals from new dataset.

Table 1: Overview of the dataset

|  |
| --- |
| Data points: 146  Features: 21  POI in Dataset: 18  All POI: 35  Stock value of James Prentice: 1095040  Wesley Colwell to POI emails: 11  Stock options of Jeffrey Skilling: 19250000  Largest total payment earner and payment: LAY KENNETH L 103559793  Salaries available: 95  Emails available: 111  NaN for total payment and percentage: 21 |

The person of interest is the person who is assumed to be committing the crime. In the given dataset, there are 18 people of interest (POI) and 128 non-POIs [[[7]](#endnote-5)]. Since some of the machine learning algorithms are successfully implemented in different industries to classification purposes, those algorithms might come handy for this type of problem domain.

## Detecting outliers and NaN values

Any form of anomalies ought to be determined and removed from the dataset before implementing any relevant machine learning algorithms. In the given dataset, there might be outlier or NAN values.

Considering the salary and bonus of the executives, it is observed that there is an outlier in the dataset, shown in Figure 1: Outlier in the dataset. The data point is for the number for total salary and bonus. This outlier data point has been removed and bonus vs salary scatter plot of the remaining dataset is being drawn in Figure 2: Salary vs bonus scatter plot after removing outlier[[8]](#footnote-3). The programming code of outlier removal process is shown in Table 2: The code of outlier removal.

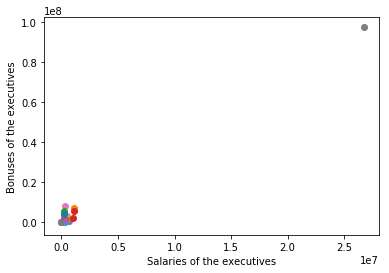


Figure 1: Outlier in the dataset

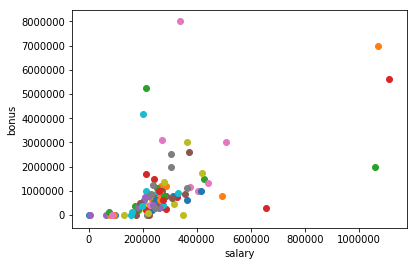


Figure 2: Salary vs bonus scatter plot after removing outlier

Table 2: The code of outlier removal

|  |
| --- |
| features = ["salary", "bonus"]  data\_dict.pop('TOTAL', 0)  data\_outlier\_removed = featureFormat(data\_dict, features)  data = data\_outlier\_removed  """  list.pop([i])  Remove the item at the given position in the list, and return it.  If no index is specified, a.pop() removes and returns the last item in the list.  The square brackets around the i in the method signature denote that the parameter is optional,  """ |

An analysis has been conducted to check if there is any NaN values exist in the dataset, & if any, they are removed. deferral\_payments, director\_fee and some other features have large number of NaN values [[[9]](#endnote-6)]. Most of the NaN values (actually all of them) have been discarded from the dataset.

From the exploration of the dataset, and relevant scenarios, it is legitimate to conclude that it is a classification problem in machine learning because determining the data point (people) as POI or Not POI is the main goal of this machine learning algorithm.

# Feature Engineering

## Introducing new features

The email sent from someone to POI would be in a great interest for this type of research. Additionally, any email that POI sent would be another significant feature for analysis. fraction\_from\_poi\_email and fraction\_to\_poi\_email are the two new features that have been introduced in the algorithm[[10]](#footnote-4).

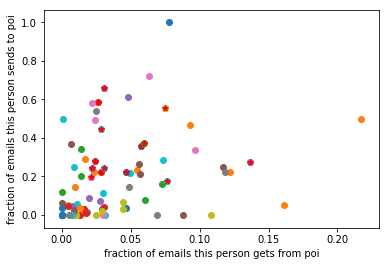


Figure 3: Fraction from and to POI



Without new features the precision and recall numbers were abysmal, with new features introduced, the performances were ameliorated into 0.31 & 0.83 respectively for precision and recall. The relevant programming code for this section is referred in [[[11]](#endnote-7)]. Therefore, new features were added into final feature sets.

## Intelligently select features

Principal Component Analysis (PCA) [[12]](#endnote-8)

After implementing PCA analysis, top five features selected are reduced into two principal components. However, meaningful information cannot be obtained from the output. The reduction of the dimensionality for first few rows in the dataset is depicted in Figure 4: Performing PCA[[13]](#footnote-7). The Prediction accuracy for the normal dataset with PCA is 66.67% and Prediction accuracy for the standardized dataset with PCA 58.33%.

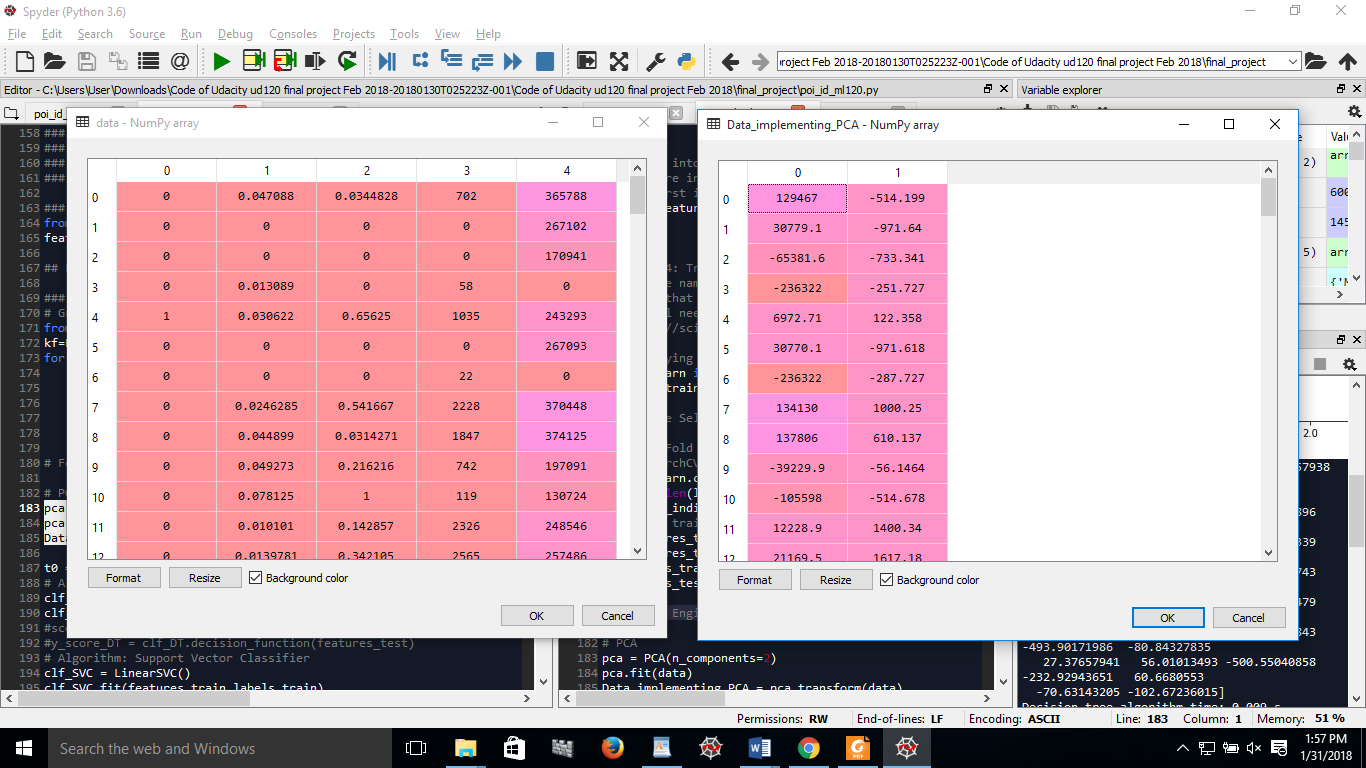


Figure 4: Performing PCA

Linear Discriminant Analysis (LDA)

LDA is being implemented; however, the performance of the algorithm seems same as before. The proper reason behind this peculiar scenario is not explored in this project. It would be legitimate to conclude that feature reduction techniques are not appropriate during the selection process of the given dataset.

## Scaling the feature

When the features can be represented more precisely, the accuracy of the algorithm will be enhanced depending on the dataset. The scaling in machine learning algorithms treats relationship between features with a correct prior assumption about the features.[[14]](#footnote-8)

In the feature engineering stage, feature scaling has been performed to the dataset. However, the performance of the algorithm is indifferent to features with or without scaling [[[15]](#endnote-9)]. In both case, the prediction accuracy was 81.04% to the PCA applied dataset. Therefore, the machine learning algorithm is implemented to the dataset without scaling.

## Selecting final features

The features which contains most prominent information about the full set of data should be selected during the feature selection process. A forest of trees from the training set (X, y) is being built, the importance of the features has been established. The final ranking of the features in the dataset (chronological importance) is documented in Table 3: Importance of the selected features.

3

|  |  |
| --- | --- |
| poi  salary  bonus  fraction\_from\_poi\_email  fraction\_to\_poi\_email | 0.161693  0.252356  0.237884  0.138795  0.209273 |

There are two new features that have been introduced in this project. Even though some intelligent feature selection methods were tried, but none of them survived in the final round. There are five prominent features that have been utilized during the modeling stage of the machine learning algorithm. The final features used in the algorithm are given below[[16]](#footnote-9):

|  |
| --- |
| features\_list = ['poi','fraction\_from\_poi\_email', 'fraction\_to\_poi\_email', 'salary', 'bonus'] [[17]](#footnote-10) |



# Machine Learning Algorithms to implement

## Pick an algorithm

Since the problem at hand is a classification problem, regression and clustering algorithms have been rejected. The dataset cannot be considered as big data, so relevant algorithms are not considered. The goal is to classify individuals as either a POI or non-POI. Since this is a classification problem, decision tree, k nearest neighbor, support vector classifier, logistic regression and adaboost learning algorithms have been implemented to the given dataset.

The precision and recall are tabulated in Table 4: Precision and Recall of different algorithms. It is detected that ***decision tree has higher precision and recall*** rate, therefore, decision tree is being selected for further analysis[[18]](#footnote-11). To sum up, precision is the number of correct positive classifications divided by the total number of positive labels assigned. It is the fraction of persons of interest predicted by the algorithm that are truly persons of interest. Recall is the number of correct positive classifications divided by the number of positive instances that should have been identified.

Table 4: Precision and Recall of different algorithms

|  |  |  |
| --- | --- | --- |
| Algorithm | Precision | Recall |
| DT | 0.534946236559 | 0.549242424242 |
| KNN | 0.4459 | 0.5 |
| SVC | 0.4310 | 0.3787 |
| LR | 0.44 | 0.484848484848 |
| AD | 0.4375 | 0.4242 |

## Tune the algorithm

The tuning is principally to find the best parameters for an algorithm to optimize its performance in given a working environment. Tuning the parameters is crucial to obtain higher performance from the algorithm for new input data[[19]](#footnote-12). Without tuning, the performance of a great algorithm would be low.

Several input parameters of the selected algorithm has been tuned manually[[20]](#footnote-13). The parameters of the algorithm have been tuned using GridSearchCV from sklearn. After tuning, the best parameters obtained are displayed in Table 5: Best parameters of Tuning the Algorithm with GridSearchCV. The precession and recall values have been improved through this tuning process as shown in Table 6: Improved precision and recall rate obtained by [[[21]](#endnote-10)].

Table 5: Best parameters of Tuning the Algorithm with GridSearchCV

|  |
| --- |
| Best parameters set found during tuning:  {'classify\_\_C': 1, 'reduce\_dim': NMF(alpha=0.0, beta\_loss='frobenius', init=None, l1\_ratio=0.0, max\_iter=200,  n\_components=1, random\_state=None, shuffle=False, solver='cd',  tol=0.0001, verbose=0), 'reduce\_dim\_\_n\_components': 1} |

Table 6: Improved precision and recall rate obtained by tuning

|  |
| --- |
| precision recall f1-score support  0.0 0.89 0.94 0.91 33  1.0 0.00 0.00 0.00 4  avg / total 0.79 0.84 0.81 37 |

# Validation and Evaluation of the algorithm

In machine learning, model validation is referred to as the process where a trained model is evaluated with a testing data set. The test data is completely new data which is not similar to training data[[22]](#footnote-14).

The common mistake is not to splitting the original dataset into training and testing dataset. If training data set is used for testing, it will over-fit, meaning that even though the model is performing amazingly during the testing phase, it will work frustratingly for new incoming dataset[[23]](#footnote-15).

The final machine learning model has been validated by ***StratifiedShuffleSplit***function written in the tester.py code given[[24]](#footnote-16).

The performance measures *recall and precision* are being utilized in this project. The precision is the probability that a person who is identified as person of interest is a true person of interest, meaning guilty person is predicted as guilty by the model developed in this project. From the Table 7: Output of the tester.py code, it is found that the accuracy of the model is 80.27%, the precision is 39%, and recall is 34%. Both precision and recall values are higher than the threshold set during the project assignment (which were 0.3 for them).

Table 7: Output of the tester.py code

|  |
| --- |
| DecisionTreeClassifier(class\_weight=None, criterion='gini', max\_depth=None,  max\_features=None, max\_leaf\_nodes=None,  min\_impurity\_decrease=0.0, min\_impurity\_split=None,  min\_samples\_leaf=1, min\_samples\_split=2,  min\_weight\_fraction\_leaf=0.0, presort=False, random\_state=None,  splitter='best')  Accuracy: 0.80450 Precision: 0.39988 Recall: 0.34550 F1: 0.37071 F2: 0.35516  Total predictions: 12000 True positives: 691 False positives: 1037 False negatives: 1309 True negatives: 8963 [[25]](#footnote-17)  Total predictions: 12000 True positives: 691 False positives: 1037 False negatives: 1309 True negatives: 8963 |

# Discussion and Conclusion

The precision can be deduced as the likelihood that a person who is identified as POI is actually a true POI; The model is 39.5% precise. Additionally, in simple English, it can be said that the model, which is developed during the project, can classify guilty person from new dataset with about 80% accuracy[[26]](#footnote-18).

***Reference***

1. http://www.geocities.com/ritholtz/writing/fiasco.html [↑](#endnote-ref-1)
2. http://money.cnn.com/2006/05/25/news/newsmakers/enron\_verdict/index.htm [↑](#endnote-ref-2)
3. *Summarize for us the goal of this project and how machine learning is useful in trying to accomplish it.* [↑](#footnote-ref-1)
4. git clone https://github.com/udacity/ud120-projects.git [↑](#endnote-ref-3)
5. exploring\_data\_ml120.py [↑](#endnote-ref-4)
6. *As part of your answer, give some background on the dataset and how it can be used to answer the project question.* [↑](#footnote-ref-2)
7. # allocation across classes (POI/non-POI)

   poi\_count = 0

   for employee in data\_dict:

   if data\_dict[employee]['poi'] == True:

   poi\_count += 1

   print ('the number of person of interest, POI = ', poi\_count)

   print ('the number of non-POI = ', len(data\_dict) - poi\_count) [↑](#endnote-ref-5)
8. *Were there any outliers in the data when you got it, and how did you handle those?* [↑](#footnote-ref-3)
9. exploring\_data\_ml120.py

   for name in enron\_data:

   if not np.isnan(float(enron\_data[name]['salary'])):

   salaries\_available += 1

   if enron\_data[name]['email\_address'] != "NaN":

   emails\_available += 1

   if np.isnan(float(enron\_data[name]['total\_payments'])):

   total\_payments\_unavailable += 1

   if enron\_data[name]['poi']:

   total\_payments\_unavailable\_poi += 1

   if np.isnan(float(enron\_data[name]['bonus'])):

   bonus\_nan += 1

   if np.isnan(float(enron\_data[name]['deferral\_payments'])):

   deferral\_payments\_nan += 1

   if np.isnan(float(enron\_data[name]['deferred\_income'])):

   deferred\_income\_nan += 1

   if np.isnan(float(enron\_data[name]['director\_fees'])):

   director\_fees\_nan += 1

   if np.isnan(float(enron\_data[name]['exercised\_stock\_options'])):

   exercised\_stock\_options\_nan += 1 [↑](#endnote-ref-6)
10. *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*  [↑](#footnote-ref-4)
11. |  |
    | --- |
    | ### Evaluate the performance of a classifier with and without new features  data\_with\_new\_feature = featureFormat(data\_dict\_with\_new\_feature,features\_list)  labels\_initial, features\_init = targetFeatureSplit(data\_initial)  labels\_with\_new\_feature, features\_newf = targetFeatureSplit(data\_with\_new\_feature)  # #  # data\_initial  # data\_with\_new\_feature  ### The performance of the Classifier (PCA) without new feature (initial dataset)  X = features\_init  y = labels\_initial  # split into a training and testing set  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=42)  # pca analysis  n\_components = 2  pca = PCA(n\_components=n\_components)  X\_train\_pca = pca.fit\_transform(X\_train,y)  X\_test\_pca = pca.fit\_transform(X\_test,y)  lr = LogisticRegression()  pipe = Pipeline([('pca', pca), ('logistic', lr)])  pipe.fit(X\_train\_pca, y\_train)  predictions = pipe.predict(X\_test\_pca)  accuracy\_without\_new\_feature = accuracy\_score(y\_test, predictions)  precision\_without\_new\_feature = precision\_score(y\_test, predictions)  recall\_without\_new\_feature = recall\_score(y\_test, predictions)  ### The performance of the Classifier (PCA) with new feature (initial dataset)  X = features\_newf  y = labels\_with\_new\_feature  # split into a training and testing set  X\_train, X\_test, y\_train, y\_test\_nf = train\_test\_split(X, y, test\_size=0.25, random\_state=42)  # pca analysis  n\_components = 2  pca = PCA(n\_components=n\_components)  X\_train\_pca = pca.fit\_transform(X\_train,y)  X\_test\_pca = pca.fit\_transform(X\_test,y)  lr = LogisticRegression()  pipe = Pipeline([('pca', pca), ('logistic', lr)])  pipe.fit(X\_train\_pca, y\_train)  predictions1 = pipe.predict(X\_test\_pca)  accuracy\_with\_new\_feature = accuracy\_score(y\_test\_nf, predictions1)  precision\_with\_new\_feature = precision\_score(y\_test\_nf, predictions1)  recall\_with\_new\_feature = recall\_score(y\_test\_nf, predictions1) |

    [↑](#endnote-ref-7)
12. pca = PCA(n\_components=2)

    pca.fit(data)

    Data\_implementing\_PCA = pca.transform(data) [↑](#endnote-ref-8)
13. PCA during feature selection [↑](#footnote-ref-7)
14. Did you have to do any scaling? Why or why not? [↑](#footnote-ref-8)
15. |  |
    | --- |
    | ### Feature Scaling  RANDOM\_STATE = 42  FIG\_SIZE = (10, 7)  # Make a train/test split using 30% test size  # Fit to data and predict using pipelined GNB and PCA.  unscaled\_clf = make\_pipeline(PCA(n\_components=2), GaussianNB())  unscaled\_clf.fit(features\_train, labels\_train)  pred\_test = unscaled\_clf.predict(features\_test)  # Fit to data and predict using pipelined scaling, GNB and PCA.  std\_clf = make\_pipeline(StandardScaler(), PCA(n\_components=2), GaussianNB())  std\_clf.fit(features\_train, labels\_train)  pred\_test\_std = std\_clf.predict(features\_test)  # Show prediction accuracies in scaled and unscaled data.  print('\nPrediction accuracy for the normal dataset with PCA')  print('{:.2%}\n'.format(metrics.accuracy\_score(labels\_test, pred\_test)))  print('\nPrediction accuracy for the standardized dataset with PCA')  print('{:.2%}\n'.format(metrics.accuracy\_score(labels\_test, pred\_test\_std))) |

    [↑](#endnote-ref-9)
16. What features did you end up using in your POI identifier, and what selection process did you use to pick them? [↑](#footnote-ref-9)
17. and if you used an automated feature selection function like SelectKBest, please report the feature scores and reason for your choice of parameter values [↑](#footnote-ref-10)
18. *What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms?* [↑](#footnote-ref-11)
19. *What does it mean to tune the parameters of an algorithm, and what can happen if you don’t do this well?* [↑](#footnote-ref-12)
20. *How did you tune the parameters of your particular algorithm? What parameters did you tune?*  [↑](#footnote-ref-13)
21. |  |
    | --- |
    | pipe = Pipeline([  ('reduce\_dim', PCA()),  ('classify', LinearSVC())  ])  N\_FEATURES\_OPTIONS = [1, 3, 5]  C\_OPTIONS = [1, 10, 100, 1000]  param\_grid = [  {  'reduce\_dim': [PCA(iterated\_power=7), NMF()],  'reduce\_dim\_\_n\_components': N\_FEATURES\_OPTIONS,  'classify\_\_C': C\_OPTIONS  },  {  'reduce\_dim': [SelectKBest(chi2)],  'reduce\_dim\_\_k': N\_FEATURES\_OPTIONS,  'classify\_\_C': C\_OPTIONS  },  ]  reducer\_labels = ['PCA', 'NMF', 'KBest(chi2)']  grid = GridSearchCV(pipe, cv=3, n\_jobs=1, param\_grid=param\_grid)  grid.fit(features\_train,labels\_train)  y\_pred\_DT\_GridSearchCV = grid.predict(features\_test)  mean\_scores = np.array(grid.cv\_results\_['mean\_test\_score'])  # scores are in the order of param\_gr`id iteration, which is alphabetical  mean\_scores = mean\_scores.reshape(len(C\_OPTIONS), -1, len(N\_FEATURES\_OPTIONS))  # select score for best C  mean\_scores = mean\_scores.max(axis=0)  print(classification\_report(labels\_test, y\_pred\_DT\_GridSearchCV))  print("Best parameters set found during tuning:")  print()  print(grid.best\_params\_)  print("End of tuning") |

    [↑](#endnote-ref-10)
22. *What is validation,* [↑](#footnote-ref-14)
23. *and what’s a classic mistake you can make if you do it wrong* [↑](#footnote-ref-15)
24. *How did you validate your analysis* [↑](#footnote-ref-16)
25. *Give at least 2 evaluation metrics and your average performance for each of them.* [↑](#footnote-ref-17)
26. *Explain an interpretation of your metrics that says something human-understandable about your algorithm’s performance* [↑](#footnote-ref-18)