P5 Introduction to Machine Learning - “Identify Fraud from Enron Email”
Final Project
Andri Rizhakov
5/13/2016
Rev. 1

Enron Submission Free-Response 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”]

The goal of this project is to predict whether a person was a Person Of Interest (“POI”) in the Enron criminal investigation or not, given said person’s feature values and some subset of known POI’s and their feature values. Machine learning is useful in trying to accomplish this because a rigorous mathematical framework is used to classify POI’s and thus can objectively decide whether someone is in one class or the other. Other options to build this classifier are non-viable: attempting to do this process by pencil/paper would be extremely tedious, and attempting to intuit the answer by a person learning from the data is not efficient, objective, nor reliable.

Beginning with the data exploration phase, the important characteristics of the data set include:

- total number of data points:
  print "total number of data points, len(data_dict):", len(data_dict)
  >>> 146
- allocation across classes (POI/non-POI):
  list_count_poi = [key for key, value in data_dict.iteritems() if (data_dict[key]['poi']==True)]
  print len(list_count_poi)
  >>> 18 # number of POI’s, and 146-18=128 number of non-POI’s
  *print "(POI/total), pre-outlier removal:", 1.0*len(list_count_poi)/(1.0*len(data_dict))*
  >>> 0.1233
  *print "(POI/non-POI), pre-outlier removal:", 1.0*len(list_count_poi)/(1.0*len(data_dict) - 1.0*len(list_count_poi))*
  >>> 0.1406
- number of features:
  print "number of features, len(data_dict['METTS MARK'].keys()):", len(data_dict['METTS MARK'].keys())
  >>> 21

- Example of features with missing values:
  #How many folks in this dataset have a quantified salary? known email address?
  list_count_quantifiedSalary = [key for key, value in data_dict.iteritems() if (data_dict[key]['salary']!='NaN')]
  print "len(list_count_quantifiedSalary):",len(list_count_quantifiedSalary)
  >>> 95
  list_count_email_address = [key for key, value in data_dict.iteritems() if (data_dict[key]['email_address']!='NaN')]
  print "len(list_count_email_address):",len(list_count_email_address)
  >>> 111
- how many features are non-NA?
  dict_summary = {}
  dict_summary2 = {}
  for feature in data_dict[data_dict.keys()[0]].keys():
  dict_summary[feature] = [key for key, value in data_dict.iteritems() if (data_dict[key][feature]!='NaN')]
  dict_summary2[feature] = len(dict_summary[feature])
  print dict_summary2
  >>> {'salary': 95, 'to_messages': 86, 'deferral_payments': 39, 'total_payments': 125, 'loan_advances': 4, 'bonus': 82, 'email_address': 111, 'restricted_stock_deferred': 18, 'total_stock_value': 126, 'shared_receipt_with_poi': 86, 'long_term_incentive': 66, 'exercised_stock_options': 102, 'from_messages': 86, 'other': 93, 'from_poi_to_this_person': 86, 'from_this_person_to_poi': 86, 'poi': 146, 'deferred_income':
  49, 'expenses': 95, 'restricted_stock': 110, 'director_fees': 17}
- Outliers?
  By visual inspection of the provided financial data (“enron61702insiderpay.pdf”), the entries
  “TOTAL” and "THE TRAVEL AGENCY IN THE PARK" were removed because this would
  have resulted in data entry error since these entries do not represent a person; the machine learning classifier would have tried to learn from these entries which would have
  produced a faulty classifier. Removal was performed simply by:
  del data_dict["TOTAL"]
  del data_dict["THE TRAVEL AGENCY IN THE PARK"]

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 does not come ready-made in the dataset

-- 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.) In your feature selection step, if you used an algorithm like a decision tree, please also give the feature importances of the features that you use, and if you used an automated feature selection function like SelectKBest, please report the feature scores and reasons for your choice of parameter values. [relevant rubric items: "create new features", "properly scale features", "intelligently select feature"]

First, features were plotted to get an idea of the spread of a feature vs POI label (yes/no). Although there were some obvious features to keep, some less obvious to keep, and some obvious to drop, all features from the exploratory phase were kept and passed on for automatic feature selection. Later, automatically selected features were compared to those that were binned in the data exploration step to make sure that those selected made sense (See A1. 1.).

Two new features were engineered and its impact on performance evaluated. Human intuition serves as the basis for the purpose of the features "fraction_from_poi" and "fraction_to_poi" - the hypothesis is that POI's send emails to other POI's at a higher rate than then general Enron (non-POI) population. Note that a test was performed on algorithm performance with and without the feature engineering step. The following summary table compares performance:

No feature addition	Accuracy: 0.88620 Recall: 0.33100	Precision: 0.64210 F1: 0.43682
Feature addition	Accuracy: 0.88607 Recall: 0.33150	Precision: 0.64058 F1: 0.43690

The features that was created did not help with the objectives (large improvement to precision and recall value from no feature addition case), so was removed for simplicity (See A1. 2. for full details).

Scaling was only used with the SVM classifier during the algorithm selection study, only because the calculation time was extremely long without scaling. It is understood that geometric considerations of the underlying mathematics, assumptions, and modeling mechanisms were probably the cause of the long run times. Additionally, without scaling, some SVM variants were not possible to run and had crashed the Python console (e.g., Line #539 of the code). Other classifier algorithms were run without scaling because calculation times were reasonable; if objectives could not be met (excessive calculation times, poor R/P metrics), then scaling would've been introduced.

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"]

The final algorithm used in classification was "KNN" (k-nearest neighbors Classifier). The other algorithms tested were "SVM" (Support Vector Machines), "GNB" (Gaussian Naive Bayes), and "DT" (Decision Tree). The algorithm performance during the cross-validation phase differed considerably, even more so when various feature selection approaches were implemented as variables as well. Figure 1 below shows the final fit models from the GridSearchCV phase and their score.

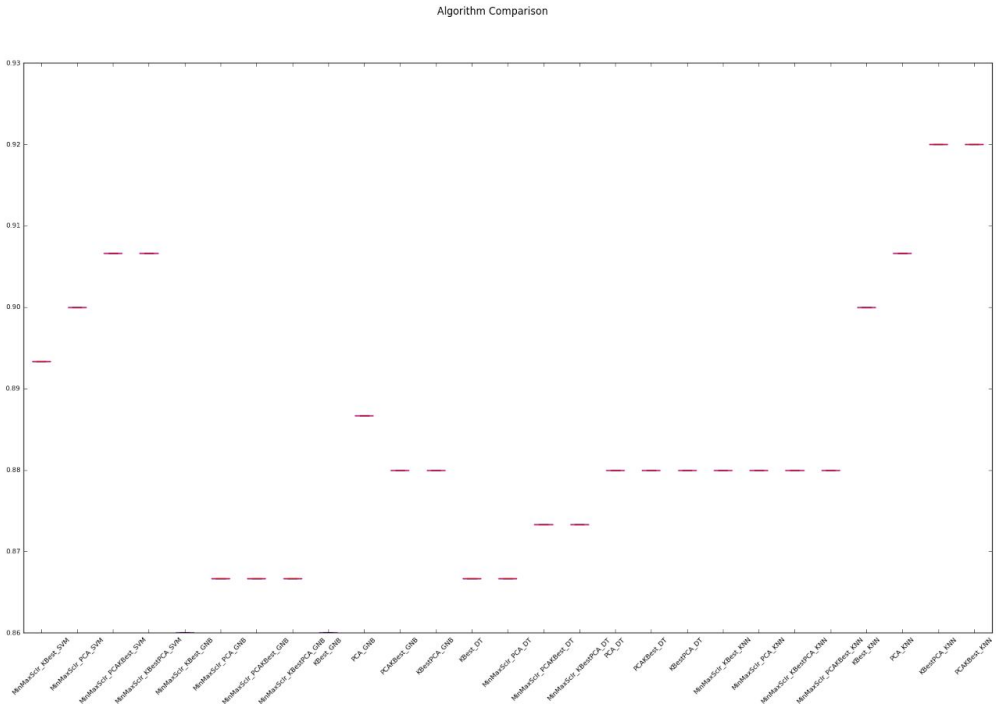


Figure 1. Comparison of various classifier algorithms during GridSearchCV phase. Note: Due to the density of possible classifiers tested, the text remains small to fit all their names; a full list can be obtained in the Appendix (A1. 16.). Default GridSearchCV metric of accuracy are the values on the y-axis. Purpose of this figure is to show the relative supremacy of the k-NN classifier (right side) over the rest of the classifier options.

Note that the default scoring method used by GridSearchCV to select a model is accuracy. k-NN was chosen because it obtained the highest accuracy (after parameter tuning). Although it is a possible shortcoming to use accuracy as the scoring method, attempts to refine this were not successful. Specifically, changing the scoring method to F1 (to combine both precision and recall metrics) did not select a viable classifier, as seen in Table 1 below.

Table 1. Final classifier results summary.

Case ID	Case description	Best clf	Best clf Pipeline steps	Metrics "train"	Metrics "test"
1	A1. 15. Full GridSearch CV. Scoring = f1. Svm max_iter=1e5. N_iter cv = 10.	"MinMaxSclr _PCAKBest_ SVM"	[('scale', MinMaxScaler(copy=True, feature_range=(0, 1))), ('features', FeatureUnion(n_jobs=1, transformer_list=[('pca', PCA(copy=True, n_components=6, whiten=False)), ('univ_select', SelectKBest(k=2, score_func=<function f_classif at 0x000000001ED1C278>))],transformer_weights=None)) , ('svm', SVC(C=1000, cache_size=200, class_weight=None, coef0=0.0, degree=3, gamma=0.9, kernel='sigmoid', max_iter=100000, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False))]	F1 = 0.563	Accuracy: 0.80653 Precision: 0.29481 Recall: 0.32400 F1: 0.30872
2	A1. 16. Full GridSearch CV. Scoring = None. Svm max_iter=1e5. N_iter cv = 10.	"KBestPCA_ KNN"	[('features', FeatureUnion(n_jobs=1, transformer_list=[('univ_select', SelectKBest(k=13, score_func=<function f_classif at 0x000000001ED1C278>)), ('pca', PCA(copy=True, n_components=1, whiten=False))], transformer_weights=None)), ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_neighbors=3, p=2, weights='uniform'))]	Accuracy = 0.920	Accuracy: 0.88620 Precision: 0.64210 Recall: 0.33100 F1: 0.43682
3	A1. 17. Full GridSearch CV. Scoring = f1. Svm max_iter=1e5. N_iter cv = 30.	"MinMaxSclr _PCA_SVM"	[('scale', MinMaxScaler(copy=True, feature_range=(0, 1))), ('features', PCA(copy=True, n_components=3, whiten=False)), ('svm', SVC(C=100, cache_size=200, class_weight=None, coef0=0.0, degree=3,	F1 = 0.421	Accuracy: 0.82200 Precision: 0.31989 Recall: 0.29750 F1: 0.30829

			gamma=100, kernel='sigmoid', max_iter=100000, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False))]		
4	A1. 18. Full GridSearch CV. Scoring = None. Svm max_iter=1e5. N_iter cv = 30.	"KBestPCA_ KNN"	[('features', FeatureUnion(n_jobs=1, transformer_list=[('univ_sele ct', SelectKBest(k=8, score_func=<function f_classif at 0x000000001ED1C278>)), ('pca', PCA(copy=True, n_components=7, whiten=False))], transformer_weights=None)), ('knn', KNeighborsClassifier(algorit hm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_neighbors=5, p=2, weights='uniform'))]	Accuracy = 0.896	Accuracy: 0.88767 Precision: 0.72597 Recall: 0.25300 F1: 0.37523

Note that the general strategy in obtaining a final precision and recall metrics of >0.3 for the performance of the classifier, was to first run an exhaustive GridSearchCV on the "training" data, and then run 'tester.py' to obtain final precision and recall values. This was preferred because copying methodology from tester.py or running tester.py on various classifier to pick the best, seemed to be a bit ineffective - it is almost as one was peeking at the test set results before making the decision on the classifier. Refining this scoring method is a possible item in the Future Work section.

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 do not 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 for the model that was not your final choice or a different model that does utilize parameter tuning, e.g. a decision tree classifier). [relevant rubric item: "tune the algorithm"]

Classifier algorithms, and general machine learning algorithms, have additional parameters that need to be specified besides the features input into the model. Unless there is knowledge about the real-world system a priori, which would help decrease uncertainty of the model by limiting the range of parameters or specifying them explicitly, parameters need to be optimized for the system to be modeled. Generally, this involves sweeping through some range of values that the parameter may take, and selecting the best performer based on some objective score metric. If

done poorly, the algorithm may be underfit or overfit; if underfit, the final algorithm has not learned from the data sufficiently and has poor precision, while overfit final algorithm takes the data too literally and cannot generalize when new data is presented to it.

The final algorithm chosen for this classifier task was tuned via GridSearchCV. Only the 'n_neighbors' parameter was chosen for recursive tuning in the "KNN" (k-nearest neighbors Classifier) algorithm.

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 is the process of checking your model to make sure that it represents the real world behavior of the system you are trying to model. In machine learning space, this is the process of splitting a data set into a training and testing portions; once the data set is trained on the training set, the performance can be checked on the fresh testing set. If the validation strategy is poor, such as testing on the trained data set, if the algorithm is totally overfit, the performance metrics that result would indicate that a perfect classifier is found (which is near impossible)!

The final "KNN" (k-nearest neighbors Classifier) algorithm was validated with StratifiedShuffleSplit routine inside the GridSearchCV routine. The performance of the feature selection and classifier algorithm set is shown in Figure 1 above. Note that StratifiedShuffleSplit was used due to the small data set (not enough to do a proper K-fold CV) and imbalanced (the ratio of POI's to the rest of the population was small, so classification is skewed towards one class in the classifier; as a result, StratifidShuffleSplit keeps the ratio of non-POI to POI the same in the training and test sets).

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"]

My final classifier algorithm (See A1. 3.) had a precision of 0.64210 and a recall of 0.33100, as reported by 'tester.py'. Recall is the ratio of true positives to true positives and false negatives; a low recall value means that the algorithm is prone to false negatives, or not predicting an actual true event. Precision is the ratio of true positives to true positives and false positives; a low precision value means that the algorithm is prone to false positives, and predicts more false alarms.

In the POI classifier exercise, it depends on which metric is seen as more important. This determination would occur in defining the objective of the problem. For example, if one wanted

to be very conservative and not make false accusations of being a POI, an algorithm with a high precision value would be desired. On the other hand, if one wanted to be very liberal and not miss any POIs (some actual POIs slip away), an algorithm with a high recall value would be desired.

Future Work

Possible future work may include:

1) Changing `scoring` parameter in GridSearchCV to a custom “F” score. Table 1 indicates that scoring based on a weighted function, that consists of a majority weight for accuracy but also includes a minor contribution for recall would help with optimizing. It appears that in Case ID 2 and 4 with greater number of GridSearchCV iterations (data set recombinations of train/test), accuracy and precision increase, but recall decreases. If recall was optimized for, perhaps precision and recall would both remain high.
2) Further research and inquiry into why the non-scaled k-NN model performed better than the scaled k-NN, which by theoretical descriptions should have been the reverse.

References

A list of Web sites, books, forums, blog posts, github repositories etc. that you referred to or used in this submission (add N/A if you did not use such resources).

I hereby confirm that this submission is my work. I have cited above the origins of any parts of the submission that were taken from Websites, books, forums, blog posts, github repositories, etc.

(Note: any other references that were not mentioned below that may have been used were either common machine learning knowledge and thus too elementary to pinpoint the source, accessible easily by searching the web, or were not mentioned by mistake. To the best of my ability, this is the complete list of references that provided significant insight and understanding towards completion of this Final Project)

[1] Udacity IML course materials (lecture notes, mini-projects, etc)

- tools/email_preprocess.py
- Enron61702insiderpay.pdf
- Email response from ‘Mike (Udacity) <review-support@udacity.com>’, titled “P5 IML "Identify Fraud from Enron Email" Final Project: Questions on Review #1”

[2] Udacity discussion forums

- https://discussions.udacity.com/t/needhelptostarttheproject/24086
- https://discussions.udacity.com/t/finalprojectfeatureselection/4621
- https://discussions.udacity.com/t/p5tyingitalltogether/35927
- https://discussions.udacity.com/t/validationandevaluation/26773
- https://discussions.udacity.com/t/confusedaboutfeatureselectionandoutliers/25018
- https://discussions.udacity.com/t/outlierremovalforsupervisedclasification/17414
- https://discussions.udacity.com/t/outlierremoval/7446

- https://discussions.udacity.com/t/outlierremovalwithaclassificationalgorithm/16573/6
- https://discussions.udacity.com/t/final-project-order-of-pipeline-operation-with-minmaxscaler-and-svc/164802

- https://discussions.udacity.com/t/final-project-clarification-needed-on-rubric-specs/164597

- https://discussions.udacity.com/t/scalercodeplacement/35653/9

- https://discussions.udacity.com/t/final-project-clarification-needed-on-rubric-specs/164597

[3] Stackoverflow sites

- https://stackoverflow.com/questions/17455302/gridsearchcv-extremely-slow-on-small-dataset-in-scikit-learn

[4] scikit-learn module documentation

- http://scikit-learn.org/stable/modules/feature_selection.html
- http://scikit-learn.org/stable/auto_examples/feature_stacker.html#example-feature-stacker-py
- http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html#sklearn.feature_selection.RFE
- http://scikit-learn.org/stable/auto_examples/feature_stacker.html

- http://scikit-learn.org/stable/modules/pipeline.html

- http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html

- http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC

- http://scikit-learn.org/stable/tutorial/statistical_inference/model_selection.html
- http://scikit-learn.org/stable/modules/generated/sklearn.cross_validation.StratifiedShuffleSplit.html
- http://scikit-learn.org/stable/modules/generated/sklearn.grid_search.GridSearchCV.html
- https://github.com/scikit-learn/scikit-learn/blob/master/examples/svm/plot_rbf_parameters.py

- http://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter

[5] Brownlee, Jason. “Machine Learning Mastery with Python” v1.0 (2016 edition).

Appendix 1. Summary of Classifier Algorithm Results

A1. 0.

Sensitivity to n_iter parameter in CV:

Cv: n_iter = 10

```
Pipeline(steps=[('features', FeatureUnion(n_jobs=1,
      transformer_list=[('pca', PCA(copy=True, n_components=1, whiten=False)), ('univ_select',
SelectKBest(k=13, score_func=<function f_classif at 0x000000001ED1F208>))],
      transformer_weights=None)), ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30,
metric='minkowski',
          metric_params=None, n_neighbors=3, p=2, weights='uniform'))])
          Accuracy: 0.88620      Precision: 0.64210      Recall: 0.33100          F1: 0.43682     F2:
0.36652
          Total predictions: 15000        True positives:  662    False positives:  369   False
negatives: 1338          True negatives: 12631

>>> best_scores
[[0.89333333333333331], [0.89333333333333331], [0.90000000000000002],
[0.85999999999999999], [0.88666666666666671], [0.88], [0.87333333333333329], [0.88],
[0.87333333333333329], [0.90000000000000002], [0.90666666666666662],
[0.92000000000000004]]

>>> grid_search.best_estimator_.steps
[('features', FeatureUnion(n_jobs=1,
      transformer_list=[('pca', PCA(copy=True, n_components=1, whiten=False)), ('univ_select',
SelectKBest(k=13, score_func=<function f_classif at 0x000000001ED1F208>))],
      transformer_weights=None)), ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30,
metric='minkowski',
          metric_params=None, n_neighbors=3, p=2, weights='uniform'))]

>>> names
['KBest_SVM', 'PCA_SVM', 'KBestPCA_SVM', 'KBest_GNB', 'PCA_GNB', 'KBestPCA_GNB',
'KBest_DT', 'PCA_DT', 'KBestPCA_DT', 'KBest_KNN', 'PCA_KNN', 'KBestPCA_KNN']
```

Runtime: ~6e3 sec

Cv: n_iter = 100

Pipeline(steps=[('features', SelectKBest(k=4, score_func=<function f_classif at 0x000000001ED1F208>)), ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
metric_params=None, n_neighbors=6, p=2, weights='uniform'))])
Accuracy: 0.88280 Precision: 0.98016 Recall: 0.12350 F1: 0.21936 F2: 0.14966
Total predictions: 15000 True positives: 247 False positives: 5 False negatives: 1753 True negatives: 12995

```
>>> best_scores
[[0.88533333333333331], [0.88066666666666671], [0.88600000000000001],
[0.84333333333333338], [0.87133333333333329], [0.85666666666666669],
[0.8666666666666667], [0.8666666666666667], [0.8666666666666667],
[0.88666666666666671], [0.8813333333333333], [0.8833333333333333]]


>>> names
['KBest_SVM', 'PCA_SVM', 'KBestPCA_SVM', 'KBest_GNB', 'PCA_GNB', 'KBestPCA_GNB',
'KBest_DT', 'PCA_DT', 'KBestPCA_DT', 'KBest_KNN', 'PCA_KNN', 'KBestPCA_KNN']

>>> best_estimators[best_scores.index(max(best_scores))]
Pipeline(steps=[('features', SelectKBest(k=4, score_func=<function f_classif at
0x000000001ED1F208>)), ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30,
metric='minkowski',
        metric_params=None, n_neighbors=6, p=2, weights='uniform'))])
```

> total run time: 82004.727 s (22 hr)

A1. 1.

Sanity check the selected KBest features

```
>>> kbestFeaturesIndices =
best_estimators[best_scores.index(max(best_scores))].named_steps["features"].get_support(in
dices=True)
>>> kbestFeaturesIndices
array([0, 4, 5, 9], dtype=int64)
>>>
>>> kBestSelectFeatureNames = [features_list[i] for i in kbestFeaturesIndices]
>>> kBestSelectFeatureNames
['poi', 'total_payments', 'exercised_stock_options', 'restricted_stock_deferred']
```

Note most features ('total_payments', 'exercised_stock_options') are consistent with the features identified in exploratory phase as possible features to use classification on.

A1. 2.

N = 10, features KBest, no new features added

```
total run time: 5736.5250001 s
>>> best_estimators[best_scores.index(max(best_scores))]
Pipeline(steps=[('features', FeatureUnion(n_jobs=1,
      transformer_list=[('pca', PCA(copy=True, n_components=1, whiten=False)), ('univ_select',
SelectKBest(k=13, score_func=<function f_classif at 0x000000001ED1F208>))],
      transformer_weights=None)), ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30,
metric='minkowski',
          metric_params=None, n_neighbors=3, p=2, weights='uniform'))])
>>> names
['KBest_SVM', 'PCA_SVM', 'KBestPCA_SVM', 'KBest_GNB', 'PCA_GNB', 'KBestPCA_GNB',
'KBest_DT', 'PCA_DT', 'KBestPCA_DT', 'KBest_KNN', 'PCA_KNN', 'KBestPCA_KNN']
>>> best_scores
[[0.89333333333333331], [0.89333333333333331], [0.90000000000000002],
[0.85999999999999999], [0.88666666666666671], [0.88], [0.87333333333333329], [0.88],
[0.88], [0.90000000000000002], [0.90666666666666662], [0.92000000000000004]]

Pipeline(steps=[('features', FeatureUnion(n_jobs=1,
      transformer_list=[('pca', PCA(copy=True, n_components=1, whiten=False)), ('univ_select',
SelectKBest(k=13, score_func=<function f_classif at 0x000000001ED1F208>))],
      transformer_weights=None)), ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30,
metric='minkowski',
          metric_params=None, n_neighbors=3, p=2, weights='uniform'))])
          Accuracy: 0.88620      Precision: 0.64210      Recall: 0.33100          F1: 0.43682     F2:
0.36652
          Total predictions: 15000         True positives:  662    False positives:  369   False
negatives: 1338          True negatives: 12631
>>> names[best_scores.index(max(best_scores))]
'KBestPCA_KNN'
```

N = 10, features KBest, new features added

```
total run time: 6936.14699984 s
>>> best_scores
```

[[0.89333333333333331], [0.89333333333333331], [0.90666666666666662],
[0.87333333333333329], [0.88666666666666671], [0.88], [0.87333333333333329], [0.88],
[0.90000000000000002], [0.90000000000000002], [0.90666666666666662],
[0.92000000000000004]]
>>> best_estimators[best_scores.index(max(best_scores))]
Pipeline(steps=[('features', FeatureUnion(n_jobs=1,
    transformer_list=[('pca', PCA(copy=True, n_components=1, whiten=False)), ('univ_select',
SelectKBest(k=15, score_func=<function f_classif at 0x000000001ED1F208>))],
    transformer_weights=None)), ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30,
metric='minkowski',
      metric_params=None, n_neighbors=3, p=2, weights='uniform'))])
>>> names[best_scores.index(max(best_scores))]
'KBestPCA_KNN'

Pipeline(steps=[('features', FeatureUnion(n_jobs=1,
    transformer_list=[('pca', PCA(copy=True, n_components=1, whiten=False)), ('univ_select',
SelectKBest(k=15, score_func=<function f_classif at 0x000000001ED1F208>))],
    transformer_weights=None)), ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30,
metric='minkowski',
      metric_params=None, n_neighbors=3, p=2, weights='uniform'))])
      Accuracy: 0.88607 Precision: 0.64058 Recall: 0.33150 F1: 0.43690 F2:
0.36691
      Total predictions: 15000 True positives: 663 False positives: 372 False
negatives: 1337 True negatives: 12628

A1. 3.

Final results of classifier algorithm.

N = 10
New feat = off
:
>>> names
['KBest_SVM', 'PCA_SVM', 'KBestPCA_SVM', 'KBest_GNB', 'PCA_GNB', 'KBestPCA_GNB',
'KBest_DT', 'PCA_DT', 'KBestPCA_DT', 'KBest_KNN', 'PCA_KNN', 'KBestPCA_KNN']
>>> best_scores
[[0.89333333333333331], [0.89333333333333331], [0.90000000000000002],
[0.85999999999999999], [0.88666666666666671], [0.88], [0.8666666666666667], [0.88],
[0.87333333333333329], [0.90000000000000002], [0.90666666666666662],
[0.92000000000000004]]

```
>>> best_scores[best_scores.index(max(best_scores))]
[0.92000000000000004]
>>> best_estimators[best_scores.index(max(best_scores))]
Pipeline(steps=[('features', FeatureUnion(n_jobs=1,
      transformer_list=[('pca', PCA(copy=True, n_components=1, whiten=False)), ('univ_select',
SelectKBest(k=13, score_func=<function f_classif at 0x000000001ED1D208>))],
      transformer_weights=None)), ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30,
metric='minkowski',
          metric_params=None, n_neighbors=3, p=2, weights='uniform'))])
```

Tester:

```
Pipeline(steps=[('features', FeatureUnion(n_jobs=1,
      transformer_list=[('pca', PCA(copy=True, n_components=1, whiten=False)), ('univ_select',
SelectKBest(k=13, score_func=<function f_classif at 0x000000001ED1D208>))],
      transformer_weights=None)), ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30,
metric='minkowski',
          metric_params=None, n_neighbors=3, p=2, weights='uniform'))])
          Accuracy: 0.88620      Precision: 0.64210      Recall: 0.33100          F1: 0.43682     F2:
0.36652
          Total predictions: 15000        True positives:  662    False positives:  369   False
negatives: 1338          True negatives: 12631
---
```

A1. 4.

Final results of classifier algorithm. Truncated code.

```
N = 10
New feature = off
:
>>> names
['KBestPCA_KNN']
>>> best_scores
[[0.92000000000000004]]
>>> best_scores[best_scores.index(max(best_scores))]
[0.92000000000000004]
>>> best_estimators[best_scores.index(max(best_scores))]
Pipeline(steps=[('features', FeatureUnion(n_jobs=1,
      transformer_list=[('pca', PCA(copy=True, n_components=1, whiten=False)), ('univ_select',
SelectKBest(k=13, score_func=<function f_classif at 0x000000001ED1D208>))],
      transformer_weights=None)), ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30,
metric='minkowski',
          metric_params=None, n_neighbors=3, p=2, weights='uniform'))])
```

Tester:
Pipeline(steps=[('features', FeatureUnion(n_jobs=1,
transformer_list=[('pca', PCA(copy=True, n_components=1, whiten=False)), ('univ_select',
SelectKBest(k=13, score_func=<function f_classif at 0x000000001ED1D208>))],
transformer_weights=None)), ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30,
metric='minkowski',
metric_params=None, n_neighbors=3, p=2, weights='uniform'))])
Accuracy: 0.88620 Precision: 0.64210 Recall: 0.33100 F1: 0.43682 F2:
0.36652
Total predictions: 15000 True positives: 662 False positives: 369 False
negatives: 1338 True negatives: 12631

A1. 5.

Final results of classifier algorithm. Truncated code. Scoring = f1

N = 10
New feature = off
:
>>> names
['KBestPCA_KNN']
>>> best_scores
[[0.51666666666666672]]
>>> best_scores[best_scores.index(max(best_scores))]
[0.51666666666666672]
>>> best_estimators[best_scores.index(max(best_scores))]
Pipeline(steps=[('features', FeatureUnion(n_jobs=1,
transformer_list=[('pca', PCA(copy=True, n_components=1, whiten=False)), ('univ_select',
SelectKBest(k=13, score_func=<function f_classif at 0x000000001ED1D208>))],
transformer_weights=None)), ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30,
metric='minkowski',
metric_params=None, n_neighbors=3, p=2, weights='uniform'))])
Tester:
Pipeline(steps=[('features', FeatureUnion(n_jobs=1,
transformer_list=[('pca', PCA(copy=True, n_components=1, whiten=False)), ('univ_select',
SelectKBest(k=13, score_func=<function f_classif at 0x000000001ED1D208>))],
transformer_weights=None)), ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30,
metric='minkowski',
metric_params=None, n_neighbors=3, p=2, weights='uniform'))])
Accuracy: 0.88620 Precision: 0.64210 Recall: 0.33100 F1: 0.43682 F2:
0.36652

Total predictions: 15000 True positives: 662 False positives: 369 False negatives: 1338 True negatives: 12631

A1. 6.

Final results of classifier algorithm. Truncated code. Scoring = none. n_jobs = 4.

N = 10
New feature = off
:
>>> names
Not defined !

total run time: 19412.7619998 s

Tester:
Pipeline(steps=[('features', FeatureUnion(n_jobs=1,
transformer_list=[('pca', PCA(copy=True, n_components=1, whiten=False)), ('univ_select', SelectKBest(k=13, score_func=<function f_classif at 0x000000001ED1C278>))],
transformer_weights=None)), ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
metric_params=None, n_neighbors=3, p=2, weights='uniform'))])
Accuracy: 0.88620 Precision: 0.64210 Recall: 0.33100 F1: 0.43682 F2: 0.36652
Total predictions: 15000 True positives: 662 False positives: 369 False negatives: 1338 True negatives: 12631

A1. 7.

Final results of classifier algorithm. Full GridSearch CV. Scoring = f1.

N = 10
New feature = off
:
total run time: 5645.48000002 s

>>> names
Names not defined! (Probably from the new main() method approach)

Tester:
Pipeline(steps=[('features', FeatureUnion(n_jobs=1,

transformer_list=[('pca', PCA(copy=True, n_components=1, whiten=False)), ('univ_select', SelectKBest(k=13, score_func=<function f_classif at 0x000000001ED1C278>))],
transformer_weights=None)), ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
metric_params=None, n_neighbors=3, p=2, weights='uniform'))])
Accuracy: 0.88620 Precision: 0.64210 Recall: 0.33100 F1: 0.43682 F2: 0.36652
Total predictions: 15000 True positives: 662 False positives: 369 False negatives: 1338 True negatives: 12631

A1. 8.

Final results of classifier algorithm. Full GridSearch CV. Scoring = recall.

N = 10
New feature = off
:
total run time: 5284.99099994 s

>>> names
Names not defined! (Probably from the new main() method approach)

Tester:
Pipeline(steps=[('features', SelectKBest(k=3, score_func=<function f_classif at 0x000000001ED4E278>)), ('scale', MinMaxScaler(copy=True, feature_range=(0, 1))), ('svm', SVC(C=10000, cache_size=200, class_weight=None, coef0=0.0, degree=3, gamma=10,
kernel='poly', max_iter=1000, probability=False, random_state=2016,
shrinking=True, tol=0.001, verbose=False))])
Accuracy: 0.39587 Precision: 0.12722 Recall: 0.60250 F1: 0.21008 F2: 0.34484
Total predictions: 15000 True positives: 1205 False positives: 8267 False negatives: 795 True negatives: 4733

A1. 9.

Final results of classifier algorithm. Full GridSearch CV. Scoring = f1.

N = 10
New feature = off
:
total run time: 6133.1099999 s

names: ['KBest_SVM', 'PCA_SVM', 'KBestPCA_SVM', 'KBest_GNB', 'PCA_GNB', 'KBestPCA_GNB', 'KBest_DT', 'PCA_DT', 'KBestPCA_DT', 'KBest_KNN', 'PCA_KNN', 'KBestPCA_KNN']

best_scores: [[0.39666666666666667], [0.54666666666666663], [0.56333333333333335], [0.35666666666666669], [0.45333333333333331], [0.47333333333333333], [0.33333333333333331], [0.30333333333333334], [0.42666666666666669], [0.23333333333333334], [0.13333333333333333], [0.26333333333333331]]

best_scores[best_scores.index(max(best_scores))]: [0.56333333333333335]

best_estimators[best_scores.index(max(best_scores))]: Pipeline(steps=[('scale', MinMaxScaler(copy=True, feature_range=(0, 1))), ('features', FeatureUnion(n_jobs=1,
      transformer_list=[('pca', PCA(copy=True, n_components=6, whiten=False)), ('univ_select', SelectKBest(k=2, score_func=<function f_classif at 0x000000001ED44278>))],
      transformer_weights=N... max_iter=10000, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False))])

KBestPCA_SVM

Tester:
Pipeline(steps=[('scale', MinMaxScaler(copy=True, feature_range=(0, 1))), ('features', FeatureUnion(n_jobs=1,
      transformer_list=[('pca', PCA(copy=True, n_components=6, whiten=False)), ('univ_select', SelectKBest(k=2, score_func=<function f_classif at 0x000000001ED44278>))],
      transformer_weights=N... max_iter=10000, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False))])
        Accuracy: 0.80653   Precision: 0.29481   Recall: 0.32400   F1: 0.30872   F2: 0.31771
        Total predictions: 15000   True positives: 648   False positives: 1550 False negatives: 1352   True negatives: 11450

A1. 10.

Final results of classifier algorithm. Full GridSearch CV. Scoring = None.

N = 10
New feature = off
:
total run time: 6025.31800008 s

names:

['KBest_SVM', 'PCA_SVM', 'KBestPCA_SVM', 'KBest_GNB', 'PCA_GNB', 'KBestPCA_GNB',
'KBest_DT', 'PCA_DT', 'KBestPCA_DT', 'KBest_KNN', 'PCA_KNN', 'KBestPCA_KNN']

best_scores:
[[0.89333333333333331], [0.90000000000000002], [0.90666666666666662],
[0.85999999999999999], [0.8666666666666667], [0.8666666666666667],
[0.87333333333333329], [0.8666666666666667], [0.87333333333333329], [0.88], [0.88], [0.88]]

names[best_scores.index(max(best_scores))]:
KBestPCA_SVM

best_scores[best_scores.index(max(best_scores))]:
[0.90666666666666662]

best_estimators[best_scores.index(max(best_scores))]:
Pipeline(steps=[('scale', MinMaxScaler(copy=True, feature_range=(0, 1))), ('features',
FeatureUnion(n_jobs=1,
      transformer_list=[('pca', PCA(copy=True, n_components=1, whiten=False)), ('univ_select',
SelectKBest(k=11, score_func=<function f_classif at 0x000000001ED44278>))],
      transformer_weights=... max_iter=10000, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False))])

best_estimators[best_scores.index(max(best_scores))].steps:
[('scale', MinMaxScaler(copy=True, feature_range=(0, 1))), ('features', FeatureUnion(n_jobs=1,
      transformer_list=[('pca', PCA(copy=True, n_components=1, whiten=False)), ('univ_select',
SelectKBest(k=11, score_func=<function f_classif at 0x000000001ED44278>))],
      transformer_weights=None)), ('svm', SVC(C=1, cache_size=200, class_weight=None,
coef0=0.0, degree=3, gamma=0.9,
  kernel='sigmoid', max_iter=10000, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False))]

Tester:
Pipeline(steps=[('scale', MinMaxScaler(copy=True, feature_range=(0, 1))), ('features',
FeatureUnion(n_jobs=1,
      transformer_list=[('pca', PCA(copy=True, n_components=1, whiten=False)), ('univ_select',
SelectKBest(k=11, score_func=<function f_classif at 0x000000001ED44278>))],
      transformer_weights=... max_iter=10000, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False))])
        Accuracy: 0.87900   Precision: 0.75910   Recall: 0.13550   F1: 0.22995   F2:
0.16214
        Total predictions: 15000   True positives: 271   False positives: 86   False
negatives: 1729   True negatives: 12914

A1. 11.

Final results of classifier algorithm. Full GridSearch CV. Scoring = None. Svm max_iter=100000.

N = 10
New feature = off
:
total run time: 8042.21099997 s

names:
['KBest_SVM', 'PCA_SVM', 'KBestPCA_SVM', 'KBest_GNB', 'PCA_GNB', 'KBestPCA_GNB', 'KBest_DT', 'PCA_DT', 'KBestPCA_DT', 'KBest_KNN', 'PCA_KNN', 'KBestPCA_KNN']

best_scores:
[[0.89333333333333331], [0.90000000000000002], [0.90666666666666662], [0.85999999999999999], [0.8666666666666667], [0.8666666666666667], [0.87333333333333329], [0.8666666666666667], [0.87333333333333329], [0.88], [0.88], [0.88]]

names[best_scores.index(max(best_scores))]:
KBestPCA_SVM

best_scores[best_scores.index(max(best_scores))]:
[0.90666666666666662]

best_estimators[best_scores.index(max(best_scores))]: Pipeline(steps=[('scale', MinMaxScaler(copy=True, feature_range=(0, 1))), ('features', FeatureUnion(n_jobs=1, transformer_list=[('pca', PCA(copy=True, n_components=1, whiten=False)), ('univ_select', SelectKBest(k=11, score_func=<function f_classif at 0x000000001ED44278>))], transformer_weights=...max_iter=100000, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False))])

best_estimators[best_scores.index(max(best_scores))].steps: [('scale', MinMaxScaler(copy=True, feature_range=(0, 1))), ('features', FeatureUnion(n_jobs=1, transformer_list=[('pca', PCA(copy=True, n_components=1, whiten=False)), ('univ_select', SelectKBest(k=11, score_func=<function f_classif at 0x000000001ED44278>))], transformer_weights=None)), ('svm', SVC(C=1, cache_size=200, class_weight=None, coef0=0.0, degree=3, gamma=0.9, kernel='sigmoid', max_iter=100000, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False))]

Tester:
Pipeline(steps=[('scale', MinMaxScaler(copy=True, feature_range=(0, 1))), ('features', FeatureUnion(n_jobs=1,
transformer_list=[('pca', PCA(copy=True, n_components=1, whiten=False)), ('univ_select', SelectKBest(k=11, score_func=<function f_classif at 0x000000001ED44278>))],
transformer_weights=...max_iter=100000, probability=False, random_state=None,
shrinking=True, tol=0.001, verbose=False))])
Accuracy: 0.87900 Precision: 0.75910 Recall: 0.13550 F1: 0.22995 F2: 0.16214
Total predictions: 15000 True positives: 271 False positives: 86 False negatives: 1729 True negatives: 12914

A1. 12.

Final results of classifier algorithm. Full GridSearch CV. Scoring = f1. Svm max_iter=100000.

N = 10
New feature = off
:
total run time: 8042.21099997 s

names:
['KBest_SVM', 'PCA_SVM', 'KBestPCA_SVM', 'KBest_GNB', 'PCA_GNB', 'KBestPCA_GNB', 'KBest_DT', 'PCA_DT', 'KBestPCA_DT', 'KBest_KNN', 'PCA_KNN', 'KBestPCA_KNN']

best_scores:
[[0.89333333333333331], [0.90000000000000002], [0.90666666666666662], [0.85999999999999999], [0.8666666666666667], [0.8666666666666667], [0.87333333333333329], [0.8666666666666667], [0.87333333333333329], [0.88], [0.88], [0.88]]

names[best_scores.index(max(best_scores))]:
KBestPCA_SVM

best_scores[best_scores.index(max(best_scores))]:
[0.90666666666666662]

best_estimators[best_scores.index(max(best_scores))]: Pipeline(steps=[('scale', MinMaxScaler(copy=True, feature_range=(0, 1))), ('features', FeatureUnion(n_jobs=1,
transformer_list=[('pca', PCA(copy=True, n_components=1, whiten=False)), ('univ_select', SelectKBest(k=11, score_func=<function f_classif at 0x000000001ED44278>))],

transformer_weights=...max_iter=100000, probability=False, random_state=None,
shrinking=True, tol=0.001, verbose=False))])

best_estimators[best_scores.index(max(best_scores))].steps: [('scale', MinMaxScaler(copy=True, feature_range=(0, 1))), ('features', FeatureUnion(n_jobs=1,
transformer_list=[('pca', PCA(copy=True, n_components=1, whiten=False)), ('univ_select', SelectKBest(k=11, score_func=<function f_classif at 0x000000001ED44278>))],
transformer_weights=None)), ('svm', SVC(C=1, cache_size=200, class_weight=None, coef0=0.0, degree=3, gamma=0.9,
kernel='sigmoid', max_iter=100000, probability=False, random_state=None,
shrinking=True, tol=0.001, verbose=False))]

Tester:
Pipeline(steps=[('scale', MinMaxScaler(copy=True, feature_range=(0, 1))), ('features', FeatureUnion(n_jobs=1,
transformer_list=[('pca', PCA(copy=True, n_components=1, whiten=False)), ('univ_select', SelectKBest(k=11, score_func=<function f_classif at 0x000000001ED44278>))],
transformer_weights=...max_iter=100000, probability=False, random_state=None,
shrinking=True, tol=0.001, verbose=False))])
Accuracy: 0.87900 Precision: 0.75910 Recall: 0.13550 F1: 0.22995 F2: 0.16214
Total predictions: 15000 True positives: 271 False positives: 86 False negatives: 1729 True negatives: 12914

A1. 13.

Final results of classifier algorithm. Full GridSearch CV. Scoring = f1. Svm max_iter=100000. N_iter cv = 30.

N = 10
New feature = off
:
Result: N/A. crash of console @ SVM w/ n_iter = 30 and Svm max_iter=100000.

A1. 14.

Final results of classifier algorithm. Full GridSearch CV. Scoring = f1. Svm max_iter=1e5. N_iter cv = 1.

New feature = off

:
Purpose: test new methodology, then increase max_iter, cv_iter if +.
Result: remove non-scaled SVM parts. Cause of freezing.

names: ['MinMaxSclr_KBest_SVM', 'MinMaxSclr_PCA_SVM', 'MinMaxSclr_PCAKBest_SVM', 'MinMaxSclr_KBestPCA_SVM', 'MinMaxSclr_KBest_GNB', 'MinMaxSclr_PCA_GNB', 'MinMaxSclr_PCAKBest_GNB', 'MinMaxSclr_KBestPCA_GNB', 'KBest_GNB', 'PCA_GNB', 'PCAKBest_GNB', 'KBestPCA_GNB', 'KBest_DT', 'MinMaxSclr_PCA_DT', 'MinMaxSclr_PCAKBest_DT', 'MinMaxSclr_KBestPCA_DT', 'PCA_DT', 'PCAKBest_DT', 'KBestPCA_DT', 'MinMaxSclr_KBest_KNN', 'MinMaxSclr_PCA_KNN', 'MinMaxSclr_KBestPCA_KNN', 'MinMaxSclr_PCAKBest_KNN', 'KBest_KNN', 'PCA_KNN', 'KBestPCA_KNN', 'PCAKBest_KNN']

best_scores: [[0.66666666666666663], [0.66666666666666663], [0.80000000000000004], [0.80000000000000004], [0.0], [0.0], [0.0], [0.0], [0.30769230769230771], [0.0], [0.30769230769230771], [0.30769230769230771], [0.0], [0.0], [0.5], [0.5], [0.66666666666666663], [0.66666666666666663], [0.66666666666666663], [0.0], [0.0], [0.0], [0.0], [0.0], [0.66666666666666663], [0.66666666666666663], [0.66666666666666663]]

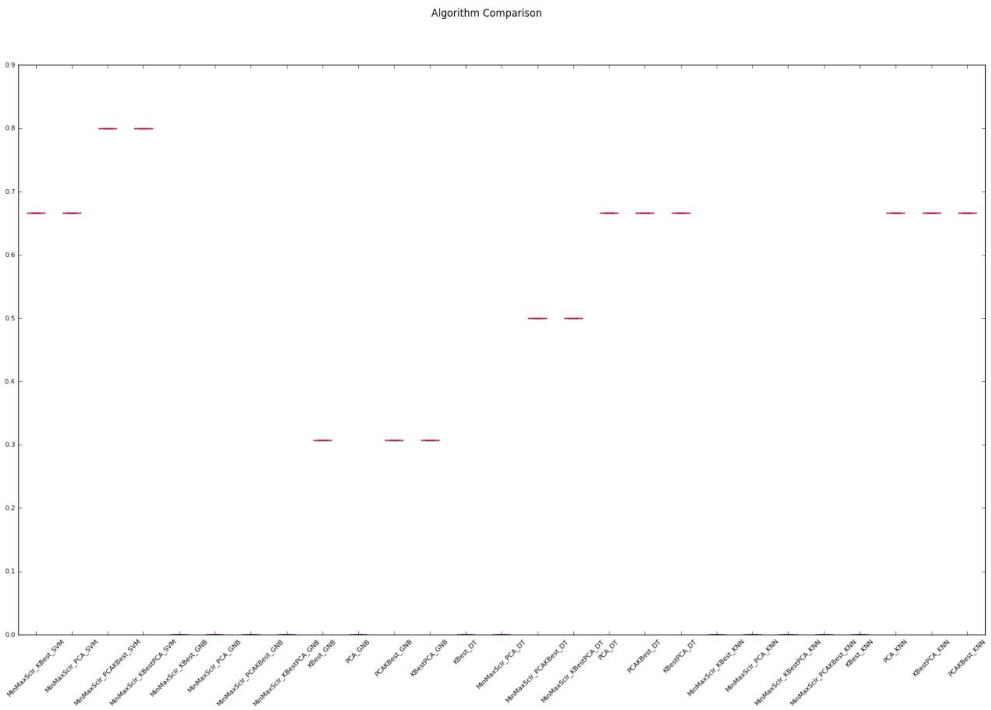
names[best_scores.index(max(best_scores))]: MinMaxSclr_PCAKBest_SVM

best_scores[best_scores.index(max(best_scores))]: [0.80000000000000004]

best_estimators[best_scores.index(max(best_scores))]: Pipeline(steps=[('scale', MinMaxScaler(copy=True, feature_range=(0, 1))), ('features', FeatureUnion(n_jobs=1, transformer_list=[('pca', PCA(copy=True, n_components=5, whiten=False)), ('univ_select', SelectKBest(k=3, score_func=<function f_classif at 0x000000001ED1C278>))], transformer_weights=N...max_iter=100000, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False))])

best_estimators[best_scores.index(max(best_scores))].steps: [('scale', MinMaxScaler(copy=True, feature_range=(0, 1))), ('features', FeatureUnion(n_jobs=1, transformer_list=[('pca', PCA(copy=True, n_components=5, whiten=False)), ('univ_select', SelectKBest(k=3, score_func=<function f_classif at 0x000000001ED1C278>))], transformer_weights=None)), ('svm', SVC(C=100, cache_size=200, class_weight=None, coef0=0.0, degree=3, gamma=10, kernel='sigmoid', max_iter=100000, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False))]

total run time: 1563.81099987 s



Tester:
Pipeline(steps=[('scale', MinMaxScaler(copy=True, feature_range=(0, 1))), ('features', FeatureUnion(n_jobs=1,
transformer_list=[('pca', PCA(copy=True, n_components=5, whiten=False)), ('univ_select', SelectKBest(k=3, score_func=<function f_classif at 0x000000001ED1C278>))],
transformer_weights=N...max_iter=100000, probability=False, random_state=None,
shrinking=True, tol=0.001, verbose=False))])
Accuracy: 0.80740 Precision: 0.25319 Recall: 0.22800 F1: 0.23994 F2: 0.23263
Total predictions: 15000 True positives: 456 False positives: 1345 False negatives: 1544 True negatives: 11655

A1. 15.

Final results of classifier algorithm. Full GridSearch CV. Scoring = f1. Svm max_iter=1e5. N_iter cv = 10.

New feature = off
:
names: ['MinMaxSclr_KBest_SVM', 'MinMaxSclr_PCA_SVM', 'MinMaxSclr_PCAKBest_SVM', 'MinMaxSclr_KBestPCA_SVM', 'MinMaxSclr_KBest_GNB', 'MinMaxSclr_PCA_GNB', 'MinMaxSclr_PCAKBest_GNB', 'MinMaxSclr_KBestPCA_GNB', 'KBest_GNB', 'PCA_GNB', 'PCAKBest_GNB', 'KBestPCA_GNB', 'KBest_DT', 'MinMaxSclr_PCA_DT',

'MinMaxSclr_PCAKBest_DT', 'MinMaxSclr_KBestPCA_DT', 'PCA_DT', 'PCAKBest_DT',
'KBestPCA_DT', 'MinMaxSclr_KBest_KNN', 'MinMaxSclr_PCA_KNN',
'MinMaxSclr_KBestPCA_KNN', 'MinMaxSclr_PCAKBest_KNN', 'KBest_KNN', 'PCA_KNN',
'KBestPCA_KNN', 'PCAKBest_KNN']

best_scores: [[0.39666666666666667], [0.54666666666666663], [0.56333333333333335],
[0.56333333333333335], [0.35666666666666669], [0.45333333333333331],
[0.47333333333333333], [0.47333333333333333], [0.35666666666666669],
[0.35666666666666669], [0.36666666666666664], [0.36666666666666664],
[0.29999999999999999], [0.28666666666666668], [0.44444444444444436],
[0.46333333333333332], [0.35333333333333333], [0.49666666666666665],
[0.43333333333333335], [0.23333333333333334], [0.13333333333333333],
[0.26333333333333331], [0.26333333333333331], [0.44], [0.46666666666666667],
[0.51666666666666672], [0.51666666666666672]]

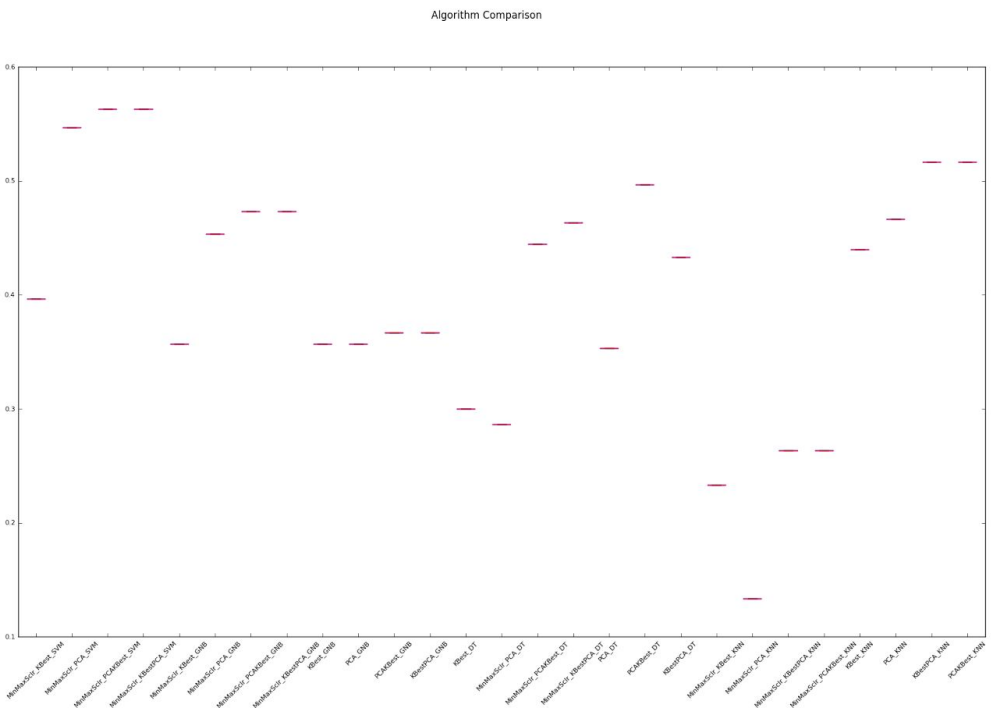
names[best_scores.index(max(best_scores))]: MinMaxSclr_PCAKBest_SVM

best_scores[best_scores.index(max(best_scores))]: [0.56333333333333335]

best_estimators[best_scores.index(max(best_scores))]: Pipeline(steps=[('scale',
MinMaxScaler(copy=True, feature_range=(0, 1))), ('features', FeatureUnion(n_jobs=1,
       transformer_list=[('pca', PCA(copy=True, n_components=6, whiten=False)), ('univ_select',
SelectKBest(k=2, score_func=<function f_classif at 0x000000001ED1C278>))],
       transformer_weights=N...max_iter=100000, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False))])

best_estimators[best_scores.index(max(best_scores))].steps: [('scale',
MinMaxScaler(copy=True, feature_range=(0, 1))), ('features', FeatureUnion(n_jobs=1,
       transformer_list=[('pca', PCA(copy=True, n_components=6, whiten=False)), ('univ_select',
SelectKBest(k=2, score_func=<function f_classif at 0x000000001ED1C278>))],
       transformer_weights=None)), ('svm', SVC(C=1000, cache_size=200, class_weight=None,
coef0=0.0, degree=3, gamma=0.9,
  kernel='sigmoid', max_iter=100000, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False))]

total run time: 16859.0999999 s



Tester:
Pipeline(steps=[('scale', MinMaxScaler(copy=True, feature_range=(0, 1))), ('features', FeatureUnion(n_jobs=1,
transformer_list=[('pca', PCA(copy=True, n_components=6, whiten=False)), ('univ_select', SelectKBest(k=2, score_func=<function f_classif at 0x000000001ED1C278>))],
transformer_weights=N...max_iter=100000, probability=False, random_state=None,
shrinking=True, tol=0.001, verbose=False))])
Accuracy: 0.80653 Precision: 0.29481 Recall: 0.32400 F1: 0.30872 F2: 0.31771
Total predictions: 15000 True positives: 648 False positives: 1550 False negatives: 1352 True negatives: 11450

A1. 16.

Final results of classifier algorithm. Full GridSearch CV. Scoring = None. Svm max_iter=1e5. N_iter cv = 10.

New feature = off
:
names: ['MinMaxSclr_KBest_SVM', 'MinMaxSclr_PCA_SVM', 'MinMaxSclr_PCAKBest_SVM', 'MinMaxSclr_KBestPCA_SVM', 'MinMaxSclr_KBest_GNB', 'MinMaxSclr_PCA_GNB', 'MinMaxSclr_PCAKBest_GNB', 'MinMaxSclr_KBestPCA_GNB', 'KBest_GNB', 'PCA_GNB', 'PCAKBest_GNB', 'KBestPCA_GNB', 'KBest_DT', 'MinMaxSclr_PCA_DT', 'MinMaxSclr_PCAKBest_DT', 'MinMaxSclr_KBestPCA_DT', 'PCA_DT', 'PCAKBest_DT',

'KBestPCA_DT', 'MinMaxSclr_KBest_KNN', 'MinMaxSclr_PCA_KNN',
'MinMaxSclr_KBestPCA_KNN', 'MinMaxSclr_PCAKBest_KNN', 'KBest_KNN', 'PCA_KNN',
'KBestPCA_KNN', 'PCAKBest_KNN']

best_scores: [[0.89333333333333331], [0.90000000000000002], [0.90666666666666662],
[0.90666666666666662], [0.85999999999999999], [0.8666666666666667],
[0.8666666666666667], [0.8666666666666667], [0.85999999999999999],
[0.88666666666666671], [0.88], [0.88], [0.8666666666666667], [0.8666666666666667],
[0.87333333333333329], [0.87333333333333329], [0.88], [0.88], [0.88], [0.88], [0.88], [0.88],
[0.88], [0.90000000000000002], [0.90666666666666662], [0.92000000000000004],
[0.92000000000000004]]

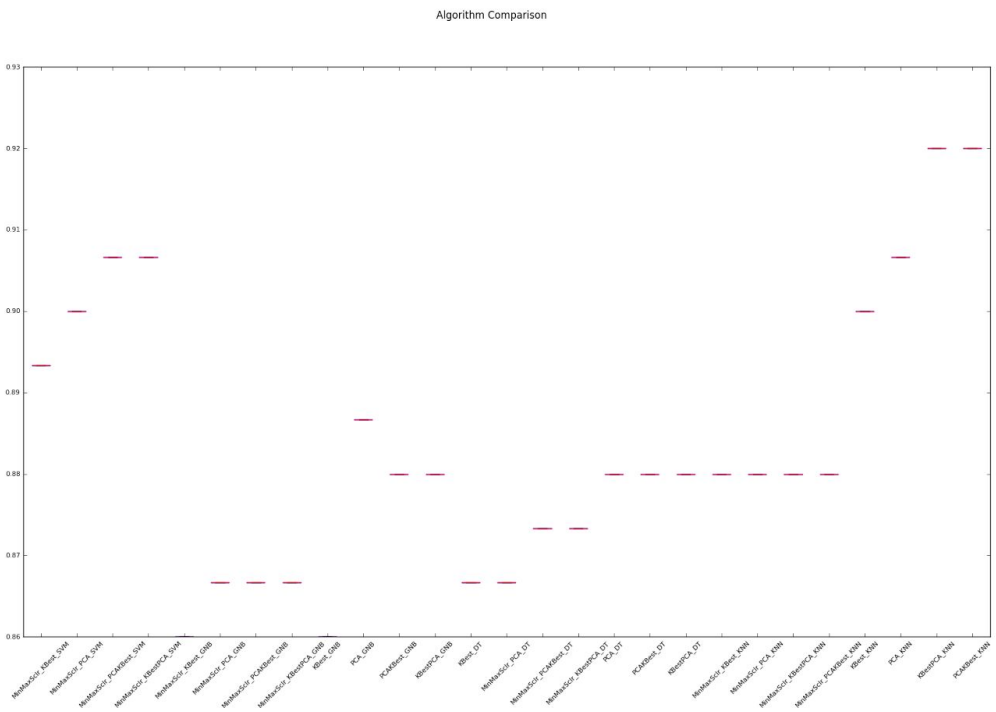
names[best_scores.index(max(best_scores))]: KBestPCA_KNN

best_scores[best_scores.index(max(best_scores))]: [0.92000000000000004]

best_estimators[best_scores.index(max(best_scores))]: Pipeline(steps=[('features',
FeatureUnion(n_jobs=1,
    transformer_list=[('univ_select', SelectKBest(k=13, score_func=<function f_classif at
0x000000001ED1C278>)), ('pca', PCA(copy=True, n_components=1, whiten=False))],
    transformer_weights=None)), ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30,
metric='minkowski',
      metric_params=None, n_neighbors=3, p=2, weights='uniform'))])

best_estimators[best_scores.index(max(best_scores))].steps: [('features',
FeatureUnion(n_jobs=1,
    transformer_list=[('univ_select', SelectKBest(k=13, score_func=<function f_classif at
0x000000001ED1C278>)), ('pca', PCA(copy=True, n_components=1, whiten=False))],
    transformer_weights=None)), ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30,
metric='minkowski',
      metric_params=None, n_neighbors=3, p=2, weights='uniform'))]

total run time: 15963.4560001 s



Tester:
Pipeline(steps=[('features', FeatureUnion(n_jobs=1,
transformer_list=[('univ_select', SelectKBest(k=13, score_func=<function f_classif at 0x000000001ED1C278>)), ('pca', PCA(copy=True, n_components=1, whiten=False))],
transformer_weights=None)), ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
metric_params=None, n_neighbors=3, p=2, weights='uniform'))])
Accuracy: 0.88620 Precision: 0.64210 Recall: 0.33100 F1: 0.43682 F2: 0.36652
Total predictions: 15000 True positives: 662 False positives: 369 False negatives: 1338 True negatives: 12631

A1. 17.

Final results of classifier algorithm. Full GridSearch CV. Scoring = f1. Svm max_iter=1e5. N_iter cv = 30.

New feature = off
:
names: ['MinMaxSclr_KBest_SVM', 'MinMaxSclr_PCA_SVM', 'MinMaxSclr_PCAKBest_SVM', 'MinMaxSclr_KBestPCA_SVM', 'MinMaxSclr_KBest_GNB', 'MinMaxSclr_PCA_GNB', 'MinMaxSclr_PCAKBest_GNB', 'MinMaxSclr_KBestPCA_GNB', 'KBest_GNB', 'PCA_GNB', 'PCAKBest_GNB', 'KBestPCA_GNB', 'KBest_DT', 'MinMaxSclr_PCA_DT', 'MinMaxSclr_PCAKBest_DT', 'MinMaxSclr_KBestPCA_DT', 'PCA_DT', 'PCAKBest_DT',

'KBestPCA_DT', 'MinMaxSclr_KBest_KNN', 'MinMaxSclr_PCA_KNN',
'MinMaxSclr_KBestPCA_KNN', 'MinMaxSclr_PCAKBest_KNN', 'KBest_KNN', 'PCA_KNN',
'KBestPCA_KNN', 'PCAKBest_KNN']

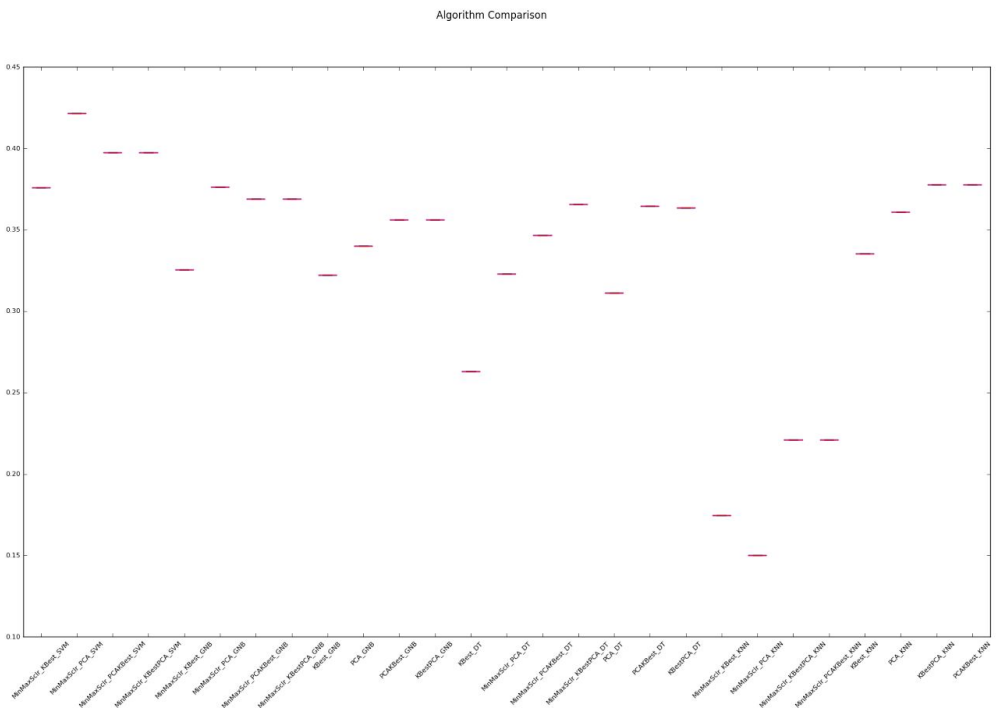
best_scores: [[0.37604978354978358], [0.4217460317460317], [0.3976638176638177],
[0.3976638176638177], [0.32545454545454544], [0.37619047619047619],
[0.36888888888888888], [0.36888888888888888], [0.32222222222222224],
[0.34000000000000002], [0.35619047619047617], [0.35619047619047617],
[0.26285714285714284], [0.32293650793650791], [0.34666666666666668],
[0.36555555555555558], [0.31111111111111112], [0.36444444444444446],
[0.36333333333333334], [0.17444444444444446], [0.14999999999999999],
[0.22111111111111112], [0.22111111111111112], [0.33555555555555555],
[0.3611111111111111], [0.37777777777777777], [0.37777777777777777]]

names[best_scores.index(max(best_scores))]: MinMaxSclr_PCA_SVM

best_scores[best_scores.index(max(best_scores))]: [0.4217460317460317]

best_estimators[best_scores.index(max(best_scores))]: Pipeline(steps=[('scale',
MinMaxScaler(copy=True, feature_range=(0, 1))), ('features', PCA(copy=True,
n_components=3, whiten=False)), ('svm', SVC(C=100, cache_size=200, class_weight=None,
coef0=0.0, degree=3, gamma=100,
  kernel='sigmoid', max_iter=100000, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False))])
best_estimators[best_scores.index(max(best_scores))].steps: [('scale',
MinMaxScaler(copy=True, feature_range=(0, 1))), ('features', PCA(copy=True,
n_components=3, whiten=False)), ('svm', SVC(C=100, cache_size=200, class_weight=None,
coef0=0.0, degree=3, gamma=100,
  kernel='sigmoid', max_iter=100000, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False))]

total run time: 53020.049 s



Tester:
Pipeline(steps=[('scale', MinMaxScaler(copy=True, feature_range=(0, 1))), ('features', PCA(copy=True, n_components=3, whiten=False)), ('svm', SVC(C=100, cache_size=200, class_weight=None, coef0=0.0, degree=3, gamma=100,
  kernel='sigmoid', max_iter=100000, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False))])
	Accuracy: 0.82200	Precision: 0.31989	Recall: 0.29750	F1: 0.30829	F2: 0.30172
	Total predictions: 15000	True positives: 595	False positives: 1265	False negatives: 1405	True negatives: 11735

A1. 18.

Final results of classifier algorithm. Full GridSearch CV. Scoring = None. Svm max_iter=1e5. N_iter cv = 30.

New feature = off
:
names: ['MinMaxSclr_KBest_SVM', 'MinMaxSclr_PCA_SVM', 'MinMaxSclr_PCAKBest_SVM', 'MinMaxSclr_KBestPCA_SVM', 'MinMaxSclr_KBest_GNB', 'MinMaxSclr_PCA_GNB', 'MinMaxSclr_PCAKBest_GNB', 'MinMaxSclr_KBestPCA_GNB', 'KBest_GNB', 'PCA_GNB', 'PCAKBest_GNB', 'KBestPCA_GNB', 'KBest_DT', 'MinMaxSclr_PCA_DT', 'MinMaxSclr_PCAKBest_DT', 'MinMaxSclr_KBestPCA_DT', 'PCA_DT', 'PCAKBest_DT', 'KBestPCA_DT', 'MinMaxSclr_KBest_KNN', 'MinMaxSclr_PCA_KNN',

'MinMaxSclr_KBestPCA_KNN', 'MinMaxSclr_PCAKBest_KNN', 'KBest_KNN', 'PCA_KNN', 'KBestPCA_KNN', 'PCAKBest_KNN']

best_scores: [[0.8866666666666671], [0.88888888888888884], [0.8866666666666671], [0.8866666666666671], [0.85111111111111115], [0.84888888888888892], [0.84222222222222221], [0.84222222222222221], [0.85111111111111115], [0.87555555555555553], [0.85777777777777775], [0.85777777777777775], [0.8666666666666667], [0.8666666666666667], [0.8666666666666667], [0.8666666666666667], [0.87333333333333329], [0.8666666666666667], [0.8666666666666667], [0.87777777777777777], [0.87333333333333329], [0.87333333333333329], [0.87333333333333329], [0.88888888888888884], [0.88888888888888884], [0.89555555555555555], [0.89555555555555555]]

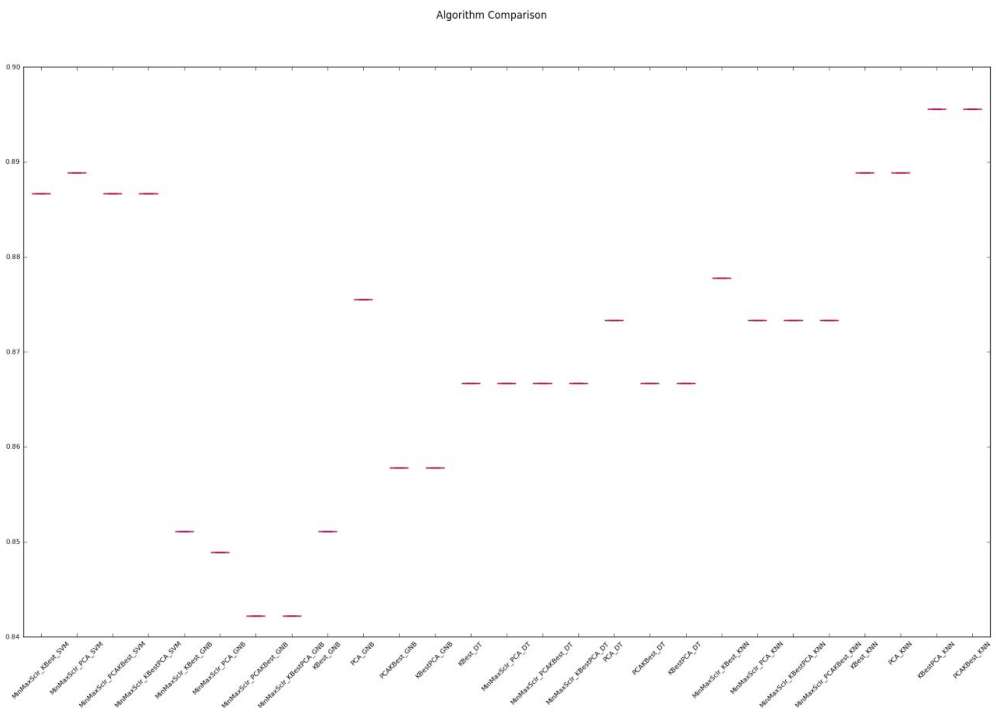
names[best_scores.index(max(best_scores))]: KBestPCA_KNN

best_scores[best_scores.index(max(best_scores))]: [0.89555555555555555]

best_estimators[best_scores.index(max(best_scores))]: Pipeline(steps=[('features', FeatureUnion(n_jobs=1,
    transformer_list=[('univ_select', SelectKBest(k=8, score_func=<function f_classif at 0x000000001ED1C278>)), ('pca', PCA(copy=True, n_components=7, whiten=False))],
    transformer_weights=None)), ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
        metric_params=None, n_neighbors=5, p=2, weights='uniform'))])

best_estimators[best_scores.index(max(best_scores))].steps: [('features', FeatureUnion(n_jobs=1,
    transformer_list=[('univ_select', SelectKBest(k=8, score_func=<function f_classif at 0x000000001ED1C278>)), ('pca', PCA(copy=True, n_components=7, whiten=False))],
    transformer_weights=None)), ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
        metric_params=None, n_neighbors=5, p=2, weights='uniform'))]

total run time: 51065.9560001 s



Tester:
Pipeline(steps=[('features', FeatureUnion(n_jobs=1,
transformer_list=[('univ_select', SelectKBest(k=8, score_func=<function f_classif at 0x000000001ED1C278>)), ('pca', PCA(copy=True, n_components=7, whiten=False))],
transformer_weights=None)), ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
metric_params=None, n_neighbors=5, p=2, weights='uniform'))])
Accuracy: 0.88767 Precision: 0.72597 Recall: 0.25300 F1: 0.37523 F2: 0.29090
Total predictions: 15000 True positives: 506 False positives: 191 False negatives: 1494 True negatives: 12809

Appendix 2. Reviewer comment resolution, from Review #1.

1. (Minor) ‘Quality of Code’: “IMPORTANT: poi_id.py takes more than an hour to run, in your next submission make sure computational time is reduced by commenting out those processes not used to generate the final classifier.”

a. Response: A flag option is added to only run the best classifier, if activated. The overall implementation is two-stepped: first, the model is run on the full GridSearch to obtain the best classifier, and then that best classifier is placed into the flagged portion of the code; this can then be run on various system to generate OS-appropriate .pkl files. This reduces run time dramatically: 6.5 mins vs. 14 hrs of the full GridSearchCV portion. Code starts on Line 769. Comment considered resolved. [Reference: Email response from 'Mike (Udacity) <review-support@udacity.com>]

2. (Major) 'Optimize Feature Selection/Engineering': "I see in your written response final classifier were tested with/without engineered feature (Well done!), however I couldn't find information explaining which feature was generated or reasons for it. Please make sure you include this info in your written response. Apart of this, results with/without engineered feature was included in your report, but a table summarizing these numbers would facilitate the readability."
    a. Response: As stated in the L11 lecture video "A New Enron Feature", the human intuition serves as the basis for the purpose of the feature - the hypothesis is that POI's send emails to other POI's at a higher rate than then general Enron (non-POI) population. This was the approach used to code up the new feature in "Task 3: Create new feature(s)" in the code, starting on Line 394. A table with the relevant results is provided for quick presentation of high level results, showing the impact of the engineered feature on the classifier performance. Comment considered resolved.
3. (Major) 'Optimize Feature Selection/Engineering': "However, my only nitpick in this case is that I miss a table in your written response with the feature scores, at this moment in your written response it is included the Feature names of the selected features but not even their scores. Please make sure you include a table showing for all features (included the engineered feature) their corresponding score."
    a. Response: `scores` can only be used when sklearn.feature_selection.SelectKBest is used as the only feature selector. For example:

```
>>> grid_search.best_estimator_.named_steps["features"].get_support(indic
es=True)
array([4, 9], dtype=int64)
>>> features_list
['poi', 'salary', 'to_messages', 'deferral_payments', 'total_payments', '
exercised_stock_options', 'bonus', 'restricted_stock', 'shared_receipt_wi
th_poi', 'restricted_stock_deferred', 'total_stock_value', 'expenses', 'l
oan_advances', 'from_messages', 'other', 'from_this_person_to_poi', 'dire
ctor_fees', 'deferred_income', 'long_term_incentive', 'from_poi_to_this_p
erson']
>>> [features_list[i] for i in grid_search.best_estimator_.named_steps["f
eatures"].get_support(indices=True)]
['total_payments', 'restricted_stock_deferred']
>>> grid_search.best_estimator_.named_steps["features"].scores_
array([ 18.28968404,   1.64634113,   0.22461127,   8.77277773,
        24.81507973,  20.79225205,   9.21281062,   8.58942073,
         0.06549965,  24.18289868,   6.09417331,   7.18405566,
         0.16970095,   4.18747751,   2.38261211,   2.1263278 ,
        11.45847658,   9.92218601,   5.24344971])
>>>
```

Note that above example is similar to the one in A1. 1.. As noted by Reviewer (comment 4 below), the first entry in the list is the 'poi' indicator or label, the remaining items in the list are the features selected. As shown in the figure above, "score_" attribute produces the selected features scores (in this example, the largest scores are the 24.8 and 24.2 values, which correspond to the "get_support" indices). Note that the above examples were used on a SelectKBest automatic selector only: if PCA is the automatic selector, "components_" and "explained_variance_ratio_" serve as similar attributes, though they cannot give a selected feature because it does not exist for PCA process. Also, if a combination of SelectKBest and PCA is used in a FeatureUnion object, it was not possible to retrieve just the SelectKBest selected features and their corresponding scores. Comment considered resolved. [Reference http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html]

4. (Major) 'Optimize Feature Selection/Engineering': "As a side comment, I would like you to check this process again:

A1. 1.

```
Sanity check the selected KBest features
>>> kbestFeaturesIndices =
best_estimators[best_scores.index(max(best_scores))].named_steps["features"].get_support(in
dices=True)
>>> kbestFeaturesIndices
array([0, 4, 5, 9], dtype=int64)
>>>
>>> kBestSelectFeatureNames = [features_list[i] for i in kbestFeaturesIndices]
>>> kBestSelectFeatureNames
['poi', 'total_payments', 'exercised_stock_options', 'restricted_stock_deferred']
```

According to feature_list poi is the first element, but note poi is not a feature but the label and can't be included in you final feature set."

   a. Response: Yes, this sanity check was performed to only serve as illustration and make sure everything was working - perhaps the variable name "kBestSelectFeatureNames" was misleading since 'poi' was selected and it is not a feature but a label. I was aware from the original poi.py template that `poi` had to be the first entry, and it was not a feature but actually a label. The output of the "Sanity check" section A1. 1. was mostly for comparing the selected features to the ones that were id'd from the Data Exploration section where potential features were plotted vs. POI status to see if a meaningful relationship existed. Comment considered resolved.
5. (Minor) 'Optimize Feature Selection/Engineering': "If algorithm calls for scaled features, feature scaling is deployed."
   a. Response: Agree with "SVM calls for scaled features not just because scaled features reduce computational costs but there are geometric considerations depending of the kernel used."; Mike Yi had a similar explanation that was used

to explain the order in the pipeline, which was a nice theoretical explanation. For completeness, poi.py was recoded to include a scaling option for all algorithms to see if performance can be improved and, additionally, make the GridSearchCV loop more readable with better organization of Selection and Scaling steps (as arrays to loop over). Note that the best performer seemed to be the k-NN classifier with non-scaled features, which seems to be slightly inconsistent with understanding obtained from lectures, Mike Yi's response, and Reviewer's suggestions - all have suggested that a geometric reason for scaling benefits for the distance-based component, but some other mechanism is allowing the non-scaled clf to perform the best. This is included in the Future Work section for further evaluation, if needed. Comment considered resolved. [Reference: Email response from 'Mike (Udacity) <review-support@udacity.com>; Udacity forums]

6. (Major) 'Pick and Tune an Algorithm': "Include a table in your written response that shows precision & recall values for your attempted algorithms."
    a. Response: Per Mike's response, better wording is used to clarify the current section. "It should be noted that the default scoring method used by GridSearchCV to select a model is accuracy. At bare minimum, you should clarify that you chose k-NN based on the fact that it obtained the highest accuracy (after parameter tuning) and that the best results of all models is noted in Figure 1." [Mike (Udacity) <reviewsupport@udacity.com>]. Text is added to elaborate on the selection strategy. Comment considered resolved. [Reference: Mike (Udacity) <reviewsupport@udacity.com>, Udacity forums]
7. (Minor) 'Pick and Tune an Algorithm': "Use the "scoring" parameter to lead the parameter search process to maximize any parameter of your choice, for example precision, recall"
    a. Response: ` scoring = 'f1' ` option is used to maximize the recall and precision because F1 parameter is a function of recall and precision, weighted evenly. See Table 1 above. Comment considered resolved. [Reference: http://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter]
8. (Minor) 'Validate and Evaluate': "Note accuracy is not a good score in this case since the dataset is unbalanced (not many POIs included), your classifier could reach high accuracy values just flagging every person as non-POI."
    a. Response: noted, but existence of this behavior was not confirmed in the model results. Comment considered resolved.
9. (Major) 'Validate and Evaluate': "Include reasons in your written response to chose stratified shuffle split in this problem over other splitting techniques available (http://scikit-learn.org/stable/modules/cross_validation.html#cross-validation-iterators). Note reasons to choose this splitting technique are related to the dataset main characteristics" "Take a look at the documentation (http://scikit-learn.org/stable/modules/generated/sklearn.cross_validation.StratifiedShuffleSplit.html#sklearn.cross_validation.StratifiedShuffleSplit)."
    a. Response: advice from Review #1 used in the cross validation explanation. Comment considered resolved. [Reference: Review #1 comments]

10. (Major) ‘Validate and Evaluate’: “The reason for marking this section off is because I wasn't able to execute poi_id.py in a decent period of time.”
    a. Response: See comment #1 above. Comment considered resolved.
11. (Minor) From “Annotations”, better visibility of plots (Title, x/y axis, etc).
    a. Response: Poor visibility of plot is addressed in the figure caption. This was made simply to show relative magnitudes of the performance of each classifier, which are indicated by the line segments, which can be seen. Comment considered resolved.
12. (Minor) From ‘Review Support’, combine all metrics into consideration for best classifier, and pick the best classifier based on this combined performance.
    a. Response: This is added as a potential follow-up step in the Future Work section. Additionally, because only one successful classifier was found, there was no way to combine with others that were not successful, because overall performance (precision and recall >0.3) would not have been achieved. Comment considered resolved.