UDACITY MACHINE LEARNING PROJECT:
IDENTIFY FRAUD FROM ENRON EMAIL

JOSEPH JASPER

11/23/2017

SECTION 1.

The purpose of this project is to leverage the principals taught to us through Udacity's Intro to Machine Learning course to create a machine learning algorithm to classify if an individual related to a company should be investigated as a person of interest for fraud. We have email and financial data left to us from the Enron investigation in the early 2000's to create an algorithm that could hopefully be leveraged for other companies in the future. Though there is a wealth of email text, there are only 146 records (18 classified as persons of interest and 128 classified as not being persons of interest).

In exploring the Enron data provided, there were some records that could be classified as outliers. There was a record for the total line of the dataset (not helpful for most cases), a record for The Travel Agency in the Park (a company not an individual) and Eugene Lockhart (who had no values associated with him). I removed these items, but there were a few individuals in the dataset that exceeded the other's by far. Unfortunately, many of these were records that belonged to high ranking individuals in the company including the CEO Ken Lay. I decided that the limited data and the fact that many of these potential outliers necessitated that I keep the records in the data set. I would hope that creating modified features, using principal component analysis or scaling many of the features would help to offset the fact that these outliers were in the dataset.

SECTION 2.

In my first submission, I had only removed the total record from the dataset. After removing the additional outliers mentioned above, the Gaussian Naïve Bayes classifier I had submitted went from a precision of .41 to .42 and from a recall of .31 to .34. I felt that the removal of 2 data points on such a small dataset was cause to restart this project. My first step was to recreate the to_poi_rate and from_poi_rate modified features and test them against the original email features. I also wanted to test the k neighbors classifier and if there would be a difference if I scaled the features. There was a modest improvement of precision (average .022) in the unscaled classifiers using the new email features. I had hoped that using the minmax scaler would make the distance feature more effective, but it did not with either the new or old email features.

I went back to reevaluate the 20 original features (not counting poi). I used Select K Best to find the right features to (hopefully) get the best results. I removed the loan advances feature, as there were only 3 records that had a value for this feature and they were all classified as poi's. The top 5 features from Select K Best were: bonus, exercised_stock_options, salary, to_poi_rate and total_stock_value. From my human intuition, this seemed reasonable as they struck me as features that could show irregularities in financial information and the users that were most potential to gain or lose based on how the company fairs as a whole. I was happy to see that one of my modified features made the cut.

K Neighbors Classifier first test results

NAME	RECALL	PRECISION
OLD EMAIL: KNEIGHBORS 3 DISTANCE	0.322	0.396
OLD EMAIL: KNEIGHBORS 3 UNIFORM	0.317	0.569
OLD EMAIL: MINMAX KNEIGHBORS 3 DISTANCE	0.232	0.394
NEW EMAIL: KNEIGHBORS 3 DISTANCE	0.327	0.416
NEW EMAIL: KNEIGHBORS 3 UNIFORM	0.319	0.593
NEW EMAIL: MINMAX KNEIGHBORS 3 DISTANCE	0.183	0.364

Old email refers to the following features: 'bonus', 'salary', 'from_poi_to_this_person', 'from_this_person_to_poi', 'from_messages', 'to_messages'

New email refers to the following features: 'bonus', 'salary', 'from_poi_rate', 'to_poi_rate'

Select K Best scores for email features sorted in descending order

FEATURE	SCORE
TO_POI_RATE	8.243
SHARED_RECEIPT_WITH_POI	5.495
FROM_POI_TO_THIS_PERSON	3.591
FROM_POI_RATE	2.349
FROM_THIS_PERSON_TO_POI	2.144
TO_MESSAGES	0.685
FROM_MESSAGES	0.171

Modified features highlighted

Select K Best scores for most available features sorted in descending order

FEATURE	SCORE
TOTAL_STOCK_VALUE	14.691
EXERCISED_STOCK_OPTIONS	13.714
SALARY	11.196
BONUS	11.129
TO_POI_RATE	8.243
RESTRICTED_STOCK	6.577
EXPENSES	5.906
SHARED_RECEIPT_WITH_POI	5.495
DEFERRED_INCOME	5.304
FROM_POI_TO_THIS_PERSON	3.591
TOTAL_PAYMENTS	2.768
LONG_TERM_INCENTIVE	2.611
FROM_POI_RATE	2.349
FROM_THIS_PERSON_TO_POI	2.144
DIRECTOR_FEES	1.796
RESTRICTED_STOCK_DEFERRED	0.763
TO_MESSAGES	0.685
DEFERRAL_PAYMENTS	0.259
FROM_MESSAGES	0.171
OTHER	0.014

Top 5 and 7 highlighted

SECTION 3.

The algorithm I finally settled on was a k neighbors classifier with 3 n_neighbors and weighted uniformly. I also used the 5 highest rated features from select k best (total_stock_value, exercised_stock_options, salary, bonus, to_poi_rate). This classifier had a precision of .62 and recall of .31.

I found this interesting graphic that I used as a general guideline to inform my choices. I had tried Naïve Bayes, SVC and k neighbors as the base classifiers and tried each with and without gridsearchcv with mixed results (posted at the end). SVC consistently underperformed in recall never reaching .07. Gaussian Naïve Bayes performed consistently well with almost every iteration I attempted having at least a .3 in both precision and recall.

SECTION 4.

Each algorithm comes with parameters that can be adjusted so that it can provide better prediction based on the specific aspects of the dataset it is being run against. Most algorithms aren't smart enough to look at the data and decide what is needed. These parameters can be so finely tuned for a specific dataset that you end up with over fitting which can hurt the algorithm's performance when predicting. However, if you are not careful with the parameters, the algorithm can be so general that it basically ignores the training data which also impact's the algorithm's usefulness.

At first, I tried to leverage a grid search algorithm to decide the best parameters for me. However, the version of grid search used on this project can't optimize for more than one score. Every time I optimized for precision, I would lose ground in recall and vice versa. Though sometimes it yielded worse results than when I tried to manually tune the parameters. Once I moved on from grid search, I reevaluated the different algorithms I had used so far. With k neighbors, I mostly tuned for k and the leaf size. With SVC, I was really interested in the differences between the kernels and the C value. Once, I finally looked at Gaussian Naïve Bayes, there weren't any parameters to tune except the number of features to receive from select k best which I tried by hand for 3, 5 and 7, I found that 5 seemed to maximize for precision while keeping recall above .3.

SECTION 5.

Validation in supervised machine learning refers to reserving a subset of the available data to test your algorithm once it has been fit to a training dataset. One obvious benefit is that it can make it easier to identify if your algorithm is overfitting for the training set. However, if you do not leverage the random_state parameter and your data is in any way sorted (especially by your classifier), you can drastically skew your results.

I initially used train_test_split, but felt I could benefit from learning additional tools. I took the recommendation from the commented-out code and leveraged StratifiedShuffleSplit. At first, I was using it incorrectly, by only pulling the train and test data from the final cycle. After looking at tester.py, I corrected my mistake and in the end received the full benefit. I move back and forth

between tester.py and the splitter on my code to evaluate. This provides 1000 separate tests of the same data, split in different ways to get the resulting scores which is truly beneficial with how small the data set is. StratifiedShuffleSplit also helps with another problem that stems from this dataset, which is that it is unbalanced. Less than 13% of the data is classified as a POI after removing the 3 outliers mentioned above. From the description in the Sklearn documentation; this method works to ensure that each fold preserves the percentage of the classes, though each fold may not be unique with this smaller dataset.

SECTION 6.

My new precision score is .62. This means that 62% percent of the time that my algorithm classified an individual as a poi, it was correct. My average recall score is .31, which means that my algorithm properly classified about 31% of the poi's when tested. Both of these scores are higher on tester.py, but that is likely because it is not holding as much data back for testing.

I believe the algorithm for this project should be optimized for precision score. This would mean that the algorithm would identify fewer of the poi's but there would be more certainty in the poi's it identified. In my mind, a project like this would be to control the scope of who is initially investigated for the sake of managing investigative resources. It is likely that investigation of one poi will identify others through investigating these leads or as the result of plea bargains, so the upfront sacrifice could be worth the long-term gains. The only concern is to make sure that the recall is not so low that the algorithm isn't identifying enough pois to be worth the effort.

APPENDIX

Run Time, Precision and Recall scores of each classifier

Name	Run Time seconds	Precision	Recall
Old Email: Kneighbors 3 distance	1.87	0.396	0.322
Old Email: Kneighbors 3 uniform	2.226	0.569	0.317
Old Email: Minmax Kneighbors 3 distance	2.316	0.394	0.232
New Email: Kneighbors 3 distance	2.024	0.416	0.327
New Email: Kneighbors 3 uniform	1.837	0.593	0.319
New Email: Minmax Kneighbors 3 distance	2.405	0.364	0.183
Old Email: Minmax Linear SVC	1.392	0	0
New Email: Minmax Linear SVC	1.308	0	0
Full feature: Gridseach Minmax SelectK SVC	824.848	0.293	0.028
Full feature: Gridseach Minmax SelectK SVC precision	878.489	0.417	0.061
Full feature: Gridseach Minmax SelectK SVC recall	878.646	0.407	0.069
Full feature: Gridseach SelectK GaussianNB	96.754	0.373	0.286
Full feature: Gridseach SelectK GaussianNB precision	108.453	0.383	0.305
Full feature: Gridseach SelectK GaussianNB recall	104.192	0.383	0.303
Full feature: Gridseach SelectK Kneighbors	1207.657	0.587	0.237
Full feature: Gridseach SelectK Kneighbors precision	1318.621	0.574	0.247
Full feature: Gridseach SelectK Kneighbors recall	1279.131	0.516	0.261
Full feature: SelectK 5 GaussianNB	2.038	0.419	0.344
Five best: Kneighbors 3 uniform	3.392	0.619	0.311
Five best: Kneighbors 5 uniform	3.322	0.642	0.282
Five best: Kneighbors 7 uniform	3.235	0.905	0.129
Seven best: Kneighbors 3 uniform	3.376	0.576	0.242
Seven best: Kneighbors 5 uniform	3.355	0.779	0.131

Feature list definitions

Old Email:

['poi', 'bonus', 'salary', 'from_poi_to_this_person', 'from_this_person_to_poi', 'from_messages', 'to_messages']

New Email:

['poi', 'bonus', 'salary', 'from_poi_rate', 'to_poi_rate']

Full Feature:

['poi', 'bonus', 'deferral_payments', 'deferred_income', 'director_fees', 'exercised_stock_options', 'expenses', 'from_messages', 'from_this_person_to_poi', 'from_poi_to_this_person', 'from_poi_rate', 'long_term_incentive', 'other', 'restricted_stock', 'restricted_stock_deferred', 'salary', 'shared_receipt_with_poi', 'to_messages', 'to_poi_rate', 'total_payments', 'total_stock_value']

Five Best:

['poi', 'total_stock_value', 'exercised_stock_options', 'salary', 'bonus', 'to_poi_rate']

Seven Best:

['poi', 'total_stock_value', 'exercised_stock_options', 'salary', 'bonus', 'to_poi_rate', 'restricted_stock', 'expenses']

Select K Best scores for used features sorted in descending order

Feature	Score
total_stock_value	14.691
exercised_stock_options	13.714
salary	11.196
bonus	11.129
to_poi_rate	8.243
restricted_stock	6.577
expenses	5.906
shared_receipt_with_poi	5.495
deferred_income	5.304
from_poi_to_this_person	3.591
total_payments	2.768
long_term_incentive	2.611
from_poi_rate	2.349
from_this_person_to_poi	2.144
director_fees	1.796
restricted_stock_deferred	0.763
to_messages	0.685
deferral_payments	0.259
from_messages	0.171
other	0.014

Classifier definitions

Name	Classifier
Old Email: Kneighbors 3 distance	KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',metric_params=None, n_jobs=1, n_neighbors=3, p=2, weights='distance')

Old Email: Kneighbors 3 uniform	KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=3, p=2, weights='uniform')
Old Email: Minmax Kneighbors 3 distance	Pipeline(steps=[('minmax', MinMaxScaler(copy=False, feature_range=(0, 1))), ('kn1', KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=3, p=2, weights='distance'))])
New Email: Kneighbors 3 distance	KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=3, p=2, weights='distance')
New Email: Kneighbors 3 uniform	KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=3, p=2, weights='uniform')
New Email: Minmax Kneighbors 3 distance	Pipeline(steps=[('minmax', MinMaxScaler(copy=False, feature_range=(0, 1))), ('kn1', KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=3, p=2, weights='distance'))])
Old Email: Minmax Linear SVC	Pipeline(steps=[('minmax', MinMaxScaler(copy=False, feature_range=(0, 1))), ('classifier', SVC(C=1.0, cache_size=200, class_weight=None, coefo=0.0, decision_function_shape=None, degree=3, gamma='auto', kernel='linear', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False))])
New Email: Minmax Linear SVC	Pipeline(steps=[('minmax', MinMaxScaler(copy=False, feature_range=(0, 1))), ('classifier', SVC(C=1.0, cache_size=200, class_weight=None, coefo=0.0, decision_function_shape=None, degree=3, gamma='auto', kernel='linear', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False))])
Full feature: Gridseach Minmax SelectK SVC	GridSearchCV(cv=None, error_score='raise', estimator=Pipeline(steps=[('minmax', MinMaxScaler(copy=False, feature_range=(0, 1))), ('feature_select', SelectKBest(k=5, score_func=<function f_classif at 0x0000000008F823C8>)), ('classifier', SVC(C=1.0, cache_size=200, class_weight=None, coefo=0.0, decision_function_shape=None, degree=3, gamma='auto', kernel='linear', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False))]), fit_params={}, iid=True, n_jobs=1, param_grid={'feature_select__k': [3, 4, 5, 6, 7], 'classifier__kernel': ['linear', 'rbf'], 'classifier__C': [1, 5, 25]}, pre_dispatch='2*n_jobs', refit=True, return_train_score=True,scoring=None, verbose=0)
Full feature: Gridseach Minmax SelectK SVC precision	GridSearchCV(cv=None, error_score='raise', estimator=Pipeline(steps=[('minmax', MinMaxScaler(copy=False, feature_range=(0, 1))), ('feature_select', SelectKBest(k=5, score_func=<function f_classif at 0x0000000008F823C8>)), ('classifier', SVC(C=1.0, cache_size=200, class_weight=None, coefo=0.0, decision_function_shape=None, degree=3, gamma='auto', kernel='linear', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False))]), fit_params={}, iid=True, n_jobs=1, param_grid={'feature_select__k': [3, 4, 5, 6, 7], 'classifier__kernel': ['linear', 'rbf'], 'classifier__C': [1, 5, 25]}, pre_dispatch='2*n_jobs', refit=True, return_train_score=True, scoring=make_scorer(precision_score), verbose=0)
Full feature: Gridseach Minmax SelectK SVC recall	GridSearchCV(cv=None, error_score='raise', estimator=Pipeline(steps=[('minmax', MinMaxScaler(copy=False, feature_range=(0, 1))), ('feature_select', SelectKBest(k=5, score_func=<function f_classif at 0x0000000008F823C8>)), ('classifier', SVC(C=1.0, cache_size=200, class_weight=None, coefo=0.0, decision_function_shape=None, degree=3, gamma='auto', kernel='linear', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False))]), fit_params={}, iid=True, n_jobs=1, param_grid={'feature_select__k': [3, 4, 5, 6, 7], 'classifier__kernel': ['linear', 'rbf'], 'classifier__C': [1, 5, 25]}, pre_dispatch='2*n_jobs', refit=True, return_train_score=True, scoring=make_scorer(recall_score), verbose=0)
Full feature: Gridseach SelectK GaussianNB	GridSearchCV(cv=None, error_score='raise', estimator=Pipeline(steps=[('feature_select', SelectKBest(k=5, score_func=<function f_classif at 0x0000000008F823C8>)), ('classifier', GaussianNB(priors=None))]),fit_params={}, iid=True, n_jobs=1, param_grid={'feature_select__k': [3, 4, 5, 6, 7]}, pre_dispatch='2*n_jobs', refit=True, return_train_score=True, scoring=None, verbose=0)

Full feature: Gridseach SelectK GaussianNB precision	GridSearchCV(cv=None, error_score='raise', estimator=Pipeline(steps=[('feature_select', SelectKBest(k=5, score_func=<function f_classif at 0x0000000008F823C8>)), ('classifier', GaussianNB(priors=None))]), fit_params={}, iid=True, n_jobs=1, param_grid={'feature_select__k': [3, 4, 5, 6, 7]}, pre_dispatch='2*n_jobs', refit=True, return_train_score=True, scoring=make_scorer(precision_score), verbose=0)
Full feature: Gridseach SelectK GaussianNB recall	GridSearchCV(cv=None, error_score='raise', estimator=Pipeline(steps=[('feature_select', SelectKBest(k=5, score_func=<function f_classif at 0x0000000008F823C8>)), ('classifier', GaussianNB(priors=None))]), fit_params={}, iid=True, n_jobs=1,param_grid={'feature_select__k': [3, 4, 5, 6, 7]}, pre_dispatch='2*n_jobs', refit=True, return_train_score=True, scoring=make_scorer(recall_score), verbose=0)
Full feature: Gridseach SelectK Kneighbors	GridSearchCV(cv=None, error_score='raise', estimator=Pipeline(steps=[('feature_select', SelectKBest(k=5, score_func=<function f_classif at 0x0000000008F823C8>)), ('classifier', KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=3, p=2, weights='distance'))]), fit_params={}, iid=True, n_jobs=1, param_grid={'feature_select__k': [3, 4, 5, 6, 7], 'classifier__n_neighbors': [3, 5, 7, 9], 'classifier__weights': ['uniform', 'distance']}, pre_dispatch='2*n_jobs', refit=True, return_train_score=True, scoring=None, verbose=0)
Full feature: Gridseach SelectK Kneighbors precision	GridSearchCV(cv=None, error_score='raise', estimator=Pipeline(steps=[('feature_select', SelectKBest(k=5, score_func=<function f_classif at 0x0000000008F823C8>)), ('classifier', KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=3, p=2, weights='distance'))]), fit_params={}, iid=True, n_jobs=1, param_grid={'feature_select__k': [3, 4, 5, 6, 7], 'classifier__n_neighbors': [3, 5, 7, 9], 'classifier__weights': ['uniform', 'distance']}, pre_dispatch='2*n_jobs', refit=True, return_train_score=True, scoring=make_scorer(precision_score), verbose=0)
Full feature: Gridseach SelectK Kneighbors recall	GridSearchCV(cv=None, error_score='raise', estimator=Pipeline(steps=[('feature_select', SelectKBest(k=5, score_func=<function f_classif at 0x0000000008F823C8>)), ('classifier', KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=3, p=2, weights='distance'))]), fit_params={}, iid=True, n_jobs=1, param_grid={'feature_select__k': [3, 4, 5, 6, 7], 'classifier__n_neighbors': [3, 5, 7, 9], 'classifier__weights': ['uniform', 'distance']}, pre_dispatch='2*n_jobs', refit=True, return_train_score=True, scoring=make_scorer(recall_score), verbose=0)
Full feature: SelectK 5 GaussianNB	Pipeline(steps=[('feature_select', SelectKBest(k=5, score_func=<function f_classif at 0x0000000008F823C8>)), ('classifier', GaussianNB(priors=None))])
Five best: Kneighbors 3 uniform	KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=3, p=2, weights='uniform')
Five best: Kneighbors 5 uniform	KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=5, p=2, weights='uniform')
Five best: Kneighbors 7 uniform	KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=7, p=2, weights='uniform')
Seven best: Kneighbors 3 uniform	KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=3, p=2, weights='uniform')
Seven best: Kneighbors 5 uniform	KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=7, p=2, weights='uniform')

Grid search best results

Name	best parameters	best estimator
Full feature: Gridseach Minmax SelectK SVC	{'feature_select__k': 3, 'classifier__kernel': 'linear', 'classifier__C': 1}	Pipeline(steps=[('minmax', MinMaxScaler(copy=False, feature_range=(0, 1))), ('feature_select', SelectKBest(k=3, score_func=<function f_classif at 0x0000000008F823C8>)), ('classifier', SVC(C=1, cache_size=200, class_weight=None, coefo=0.0, decision_function_shape=None, degree=3, gamma='auto', kernel='linear', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False))])
Full feature: Gridseach Minmax SelectK SVC precision	{'feature_select__k': 5, 'classifier__kernel': 'linear', 'classifier__C': 25}	Pipeline(steps=[('minmax', MinMaxScaler(copy=False, feature_range=(0, 1))), ('feature_select', SelectKBest(k=5, score_func=<function f_classif at 0x0000000008F823C8>)), ('classifier', SVC(C=25, cache_size=200, class_weight=None, coefo=0.0, decision_function_shape=None, degree=3, gamma='auto', kernel='linear', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False))])
Full feature: Gridseach Minmax SelectK SVC recall	{'feature_select__k': 7, 'classifier__kernel': 'linear', 'classifier__C': 5}	Pipeline(steps=[('minmax', MinMaxScaler(copy=False, feature_range=(0, 1))), ('feature_select', SelectKBest(k=7, score_func=<function f_classif at 0x0000000008F823C8>)), ('classifier', SVC(C=5, cache_size=200, class_weight=None, coefo=0.0, decision_function_shape=None, degree=3, gamma='auto', kernel='linear', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False))])
Full feature: Gridseach SelectK GaussianNB	{'feature_select__k': 5}	Pipeline(steps=[('feature_select', SelectKBest(k=5, score_func=<function f_classif at 0x0000000008F823C8>)), ('classifier', GaussianNB(priors=None))])
Full feature: Gridseach SelectK GaussianNB precision	{'feature_select__k': 5}	Pipeline(steps=[('feature_select', SelectKBest(k=5, score_func=<function f_classif at 0x0000000008F823C8>)), ('classifier', GaussianNB(priors=None))])
Full feature: Gridseach SelectK GaussianNB recall	{'feature_select__k': 6}	Pipeline(steps=[('feature_select', SelectKBest(k=6, score_func=<function f_classif at 0x0000000008F823C8>)), ('classifier', GaussianNB(priors=None))])
Full feature: Gridseach SelectK Kneighbors	{'feature_select__k': 3, 'classifier__n_neighbors': 9, 'classifier__weights': 'uniform'}	Pipeline(steps=[('feature_select', SelectKBest(k=3, score_func=<function f_classif at 0x0000000008F823C8>)), ('classifier', KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=9, p=2, weights='uniform'))])
Full feature: Gridseach SelectK Kneighbors precision	{'feature_select__k': 3, 'classifier__n_neighbors': 5, 'classifier__weights': 'uniform'}	Pipeline(steps=[('feature_select', SelectKBest(k=3, score_func=<function f_classif at 0x0000000008F823C8>)), ('classifier', KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=5, p=2, weights='uniform'))])
Full feature: Gridseach SelectK Kneighbors recall	{'feature_select__k': 3, 'classifier__n_neighbors': 3, 'classifier__weights': 'distance'}	Pipeline(steps=[('feature_select', SelectKBest(k=3, score_func=<function f_classif at 0x0000000008F823C8>)), ('classifier', KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=3, p=2, weights='distance'))])