DataAnalysis NanoDegree

Machine Learning Project by Teresa Aysan

March 5, 2018

DOCUMENTATION OF MY WORK

I hereby confirm that this submission is my work. I have cited in my "Text File Listing My References" document the origins of any parts of the submission that were taken from websites, books, forums, blog posts, github repositories, etc. I have put superscript numbers in this document and in my code to cross-reference to the origin in my "Text File Listing My References" document. Any direct quotes are in quotations and in italics. Any of my own words within the direct quotes are in normal font and in <>. The superscript numbers may not be in order. Some numbers may be missing because I no longer needed them.

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 *"put* <my> *machine learning* <(ML)> *skills to use by building an algorithm to identify Enron Employees who may have committed fraud based on the public Enron financial and email dataset.*< ML will be **useful** to accomplish this because it is through ML that I will> *extract and identify useful features that best represent* <my> *data;* <use> *a few of the most commonly used machine learning algorithms today* <to create results to answer my question>*, and ... evaluate the performance of* <my ML> *algorithms...* <Udacity> *combined the Enron email and financial data into a **dictionary**, where each key-value pair in the dictionary corresponds to one person. The dictionary key is the person's name, and the value is another dictionary, which contains the names of all the features and their values for that person. The features in the data fall into three major types, namely financial features, email features and POI labels."*[1]

My project question was: How does the financial data that is provided help me to identify POIs? Specifically, is a POI more likely to receive "loan_advances", and are the "loan_advances" tied to "deferral_payments"? After the initial investigation, it was necessary to remove my "loan_advances" feature due to missing data and to add the "bonus" feature instead. I used the dataset to answer my project question by filtering out the relevant data from the dictionary for each person, building my own feature on the ratio between my features, building a color feature to color the dots in my scatter plots, and using ML to predict, evaluate and validate if there is a relationship.

The use of the "deferral_payments" and "bonus" features did not allow me to pass the tester.py to attain a precision and recall score of at least 0.3 each. As a result, I learned that it is not best to rely on my own intuition of which features would make for a good analysis. In the end, in task 6, I used SelectKBest and GridSearch to pick the best features, which ended up being "total_stock_value", "exercised_stock_options" and "salary". The use of these 3 features allowed my code to pass tester.py minimum requirements.

Important Characteristics of the Dataset

Total Number of Datapoints: There were 146 records (people) in the original dataset. The key to one of those records was "Total", so I removed that record, for a net of 145.

Allocation across Classes POI and non-POI: There were 18 POIs in the Enron dataset. This number seemed quite low, so I compared it with the number of records and POIs in the "Names" dataset that had been given in the course, and calculated the ratio of POIs to total people. As it turned out, the Names dataset had only 4.7% whereas the Enron dataset had 12.4%.

Number of Features Used and Missing Values: In tasks 1-5, of the 21 features, I initially chose to work with two features. I chose "loan_advances" and "deferral_payments". I performed some data enquiries on my two features and found "Total" which would have been an outlier, had I left it in. Only two people, both non-POIs, had "loan_advances", so I decided that the feature, "loan_advances" had too much missing data and I switched to the "bonus" feature instead. I decided to eliminate any records that had missing data for both "deferral_payments" and "bonus".

I used scatter plots of gradual complexity to help me analyze my data. I started with a single color, then used different colors for POIs and non-POIs.

I ended up with 97 records in my_dataset. I created reporting tables within my code to ensure that I had a high enough representation of POIs, to get good accuracy. [3] "More data is always better."[3]

Description	Names File	Enron Data	My Data
Number of people:	746	145	97
Number of POIs	35	18	17
Ratio of POIs to number of people:	0.047	0.124	0.175

In task 6 when I finally realized that I should be using SelectKBest and GridSearch to select my features, I discovered that it was best to use the full dataset and that "THE TRAVEL AGENCY IN THE PARK" was an outlier that needed to be excluded. I also learned that 3 features were required to attain precision and recall scores greater than 0.3.

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", "intelligently select features", "properly scale features"]

I chose two features that I thought would indicate if the person was a POI (Person of Interest). I did not want to use "salary" because the lessons used "salary". The course said to keep it simple, so I decided to only have 2 features. As indicated in the answer to the previous question, I initially selected "deferral_payments" and "loan_advances", but when I looked at the number of NaNs, I realized that there was not enough data in the "loan_advances", so I switched that one to "bonus".

I scaled the features but it did not help. I learned that I did not have to scale to convert the numeric data to have consistent measurements, because they all had the same $ value types. However, I eliminated the records that had NaN values for both of my features.

I created a feature for the ratio between "deferral_payments" and "bonus" because I was interested to see if there was a relationship. There was not. I performed some regression testing on "deferral_payments" and "bonus". The regression line went up very very slightly to the right with a slope of 0.046. There is practically no correlation between "deferral_payments" and "bonus".

I also created a feature for the color of the points on the graphs to indicate if the point was for a POI or not. I used the color feature for graphing to help me analyze my data.

Initially I did not use PCA or SelectKBest to select my features because I did not think there were that many features to choose from that would allow me to create financial ratios. How wrong I was.

When running my "gut-feel" selections through tester.py, I could not get precision and recall higher than 0.3, so I reevaluated my approach, and realized that it is best to let SelectKBest pick the features and to let pipeline and GridSearch pick the best combination of classifier parameters.

In my definition "task6FromScratch":

- I included all of the records in the Enron database except TOTAL and the outlier "THE TRAVEL AGENCY IN THE PARK".
- I included all of the features in the Enron database except for email_addresses. I tried adding 3 of my own features, two of which were "scaled_deferral_payments" and "scaled_bonus" and the third was the ratio of "deferral_payments" to "bonus". However, when I tested tester.py with those features included, the precision and recall scores were less than 0.3, so I excluded those features in my list of final features.
- I chose to test GaussianNB and DecisionTree as two classifiers to use because they had better results when I used them in my tasks 1-5 code with my "deferral_payments" and "bonus" features.
- These are the results using GaussianNB and DecisionTree classifiers with SelectKBest and GridSearch:

GaussianNB does not have parameters to run through GridSearch, but I ran SelectKBest through GridSearch to find the best features to get to precision and recall >0.3. Initially I had used kbest with a range of (1,4), but that only generated 3 features and did not get to the required precision and recall. So, I increased kbest range to (1,10), and GuassianNB selected 5 features with precision and recall higher than 0.3. Here is the Tester Classification report:

```
Pipeline(steps=[('scaler', MinMaxScaler(copy=True, feature_range=(0, 1))), ('kbest', SelectKBest(k=5, score_func=<function f_classif at 0x000000000ECF74A8>)), ('clfNB', GaussianNB(priors=None))])
Accuracy: 0.84833	Precision: 0.41964	Recall: 0.35900	F1: 0.38696	F2: 0.36968
Total predictions: 15000	True positives:  718	False positives:  993	False negatives: 1282	True negatives: 12007
```

Here are the features selected for GaussianNB, and the score / pvalue for each: [('total_stock_value', '14.69', '0.000'), ('exercised_stock_options', '13.71', '0.000'), ('salary', '11.20', '0.001'), ('bonus', '11.13', '0.001'), ('restricted_stock', '6.58', '0.012')]

DecisionTree has parameters that I ran through GridSearch. I selected the parameters based on the selections in the forum. I also ran SelectKBest through GridSearch to find the best features to get to precision and recall >0.3. I used kbest ranges of (1,4) and (1,10). I found that both of those ranges selected the same three features with the same results, so I set kbest to a range of (1,4) so that GridSearch would take less time to process. Here is the Tester Classification report which clearly shows that precision and recall are greater than 0.3 as required by the final project evaluation:

```
Pipeline(steps=[('scaler', MinMaxScaler(copy=True, feature_range=(0, 1))), ('kbest', SelectKBest(k=3, score_func=<function f_classif at 0x000000000ECF74A8>)), ('clfTree', DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=10, max_features=None, max_leaf_nodes=30, min_impurity_split=1e-07, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=42, splitter='random'))])
Accuracy: 0.83680	Precision: 0.38297	Recall: 0.36650	F1: 0.37455	F2: 0.36968
Total predictions: 15000	True positives:  733	False positives: 1181	False negatives: 1267	True negatives: 11819
```

Here are the features selected, and the score / pvalue for each: [('total_stock_value', '14.69', '0.000'), ('exercised_stock_options', '13.71', '0.000'), ('salary', '11.20', '0.001')]

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"]*

Before task 6, using "deferral_payments" and "bonus" as my features, I tried GaussianNB, Decision Tree, RandomForest and SVM. These are the parameters that I used:

- GaussianNB(),
- DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
      max_features=None, max_leaf_nodes=None,
      min_impurity_split=1e-07, min_samples_leaf=1,
      min_samples_split=2, min_weight_fraction_leaf=0.0,
      presort=False, random_state=42, splitter='best'),
- RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
      max_depth=None, max_features='auto', max_leaf_nodes=None,
      min_impurity_split=1e-07, min_samples_leaf=1,
      min_samples_split=2, min_weight_fraction_leaf=0.0,
      n_estimators=10, n_jobs=1, oob_score=False, random_state=None,
      verbose=0, warm_start=False),
- SVC(kernel="linear")

I created variables and a loop so that I could loop through the classifiers and prepare statistical data. This allowed me to create a report of the results in a comparative table. I also showed the names file statistics and my total data set statistics in the table for comparative purposes.

I created a general dictionary to keep track of the variables by classifier, my data and anything else that I needed for my code or for my graphs. I did this in lieu of using global variables.

sklearn cross_validation

I created a definition called "def basicTrainTestData(data, labels, testPct, randomSeed, svcKernel, svcC)". In this definition, I used sklearn cross_validation to select training and testing data. I tried a number of different values in the def.

Here is a report of the results when I used the following values: testPct = 0.3, randomSeed = 42, svcKernel = "linear", svcC = 1.

SVC was very slow when using sklearn cross_validation as the tool to create test and train data, so I excluded it when running this definition, but included it in my "splitting the data manually" code which I describe below.

This report, when using sklearn cross_validation, shows that "accuracy" and "score" are perfect (1.0) in some cases. However, I do not trust the results because there are so few POIs in the Enron dataset.

######## Table of Statistics from the Basic Classifiers using sklearn cross_validation ########

Description	Names File	Enron Data	My Data	GaussianNB	Tree	Forest
Number of people:	746	145	97	97	97	97
Number of POIs:	35	18	17	1	7	7
Ratio of POIs to number of people (note 1):	0.047	0.124	0.175	0.033	0.233	0.233
Ratio of predicted POIs ratio to total ratio:	N/A	N/A	N/A	0.190	1.331	1.331

Accuracy (Note 2):	N/A	N/A	N/A	0.733	1.000	1.000
Score:	N/A	N/A	N/A	0.733	1.000	1.000
Training time to fit:	N/A	N/A	N/A	0.001	0.000	0.023
Training time to predict:	N/A	N/A	N/A	0.000	0.000	0.006

Note 1: In the case of the classifiers, the ratio is ratio of # of POIs to # of people in the test dataset.

Note 2: The accuracy and "score" are perfect for the DecisionTree and Forest classifiers.

Splitting the data manually:

I created a definition called "def complexTrainTestData(data, randomSeed, testPct, errorPct, maxDeferral, maxBonus, splitPct)" where I could adjust the def variables to create the training and test data through split to try and observe the results for various values in order to get better results. This definition code did not include sklearn cross_validation.

These are the values used for the table of results below: Random Seed = 42 Test Percent = 0.3 Error Percent = 0.1 Max Deferral Payments = 0.8 Max Bonus = 0.8 Split Percent = 0.75.

######### Complex: Table of Statistics from the Classifiers on data created by Split #########

Description	Names	Enron	My Data	NB	Tree	Forest	SVC
Number of people:	746	145	97	97	97	97	97
Number of POIs:	35	18	17	1	11	13	15
Ratio of POIs to number of people (note 1):	0.047	0.124	0.175	0.560	0.440	0.520	0.600
Ratio of predicted POIs ratio to total ratio:	N/A	N/A	N/A	3.195	2.511	2.967	3.424
Accuracy (Note 2):	N/A	N/A	N/A	0.920	0.800	0.800	0.880
Score:	N/A	N/A	N/A	0.920	0.800	0.800	0.880
Training time to fit:	N/A	N/A	N/A	0.001	0.000	0.026	0.001
Training time to predict:	N/A	N/A	N/A	0.000	0.000	0.006	0.000

Note 1: In the case of the classifiers, the ratio is ratio of # of POIs to # of people in the test dataset.

Note 2: The accuracy is highest for NB at 0.92, and SVC is a high second at 0.88.

Note 3: By using split instead of sklearn cross_validation; I have more control over the split of testing and training data. The numbers seem more reliable, since there are no perfect 1.0 metrics.

Task6

When I finally buckled down and allowed SelectKBest, pipeline and GridSearch to select the features and algorithm parameters, I got much better results.

- I tried GaussianNB with no parameters, since it does not have any, but with SelectKBest to pick the best features. I set kbest at a range of (1,10), to select enough features (5) to get precision and recall > 0.3:
- I also tried DecisionTree with the following parameter options, allowing GridSearch to pick the best; and I allowed SelectKBest to pick the best features. I found that kbest only needed a range of (1,4), anything more than that still only picked 3 features for the best results:

```
parameters = {'clfTree__criterion': ['gini','entropy'],
          'clfTree__splitter':['best','random'],
          'clfTree__min_samples_split':[2, 10, 20],
           'clfTree__max_depth':[10,15,20,25,30],
           'clfTree__max_leaf_nodes':[5,10,30],
          'clfTree__random_state': [42],
          'kbest__k': range(1,4) }
```

The results of those selections are in the answer under question 2.

The best result

- for precision was 0.41964 when using GuassianNB with features 'total_stock_value', 'exercised_stock_options', 'salary', 'bonus', and 'restricted_stock'.
- for recall was 0.36650 when using DecisionTree
    - with parameters class_weight=None, criterion='entropy', max_depth=10, max_features=None, max_leaf_nodes=30, min_impurity_split=1e-07, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=42, splitter='random'
    - and with features 'total_stock_value', 'exercised_stock_options' and 'salary'.

DecisionTree was used in the final submission to the tester.py, with the following results: **Precision: 0.38297 and Recall: 0.36650**.

4. *What does it mean to tune the parameters of an algorithm, and what can happen if you don't do this well? How did you tune the parameters of your particular algorithm? What parameters did you tune? (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 items: "discuss parameter tuning", "tune the algorithm"]*

The purpose of tuning the parameters of an algorithm is to select the best values, such that the best results are achieved for the desired output, within an optimum period of time[42]. If the results improve, then the parameters' values are more favorable, or "tuned". If the results get worse, then the parameters are unfavorable, or "untuned".

- If tuning is not done well, then it can take longer to generate the model, it can take longer to run the model, and there is either too little information for accurate predictions, or too much data that does not contribute to improvement in the overall information.
- If there is too little tuning, then the model may stop short of getting the optimum performance metrics or optimum prediction of results. The model may not consider features that are highly correlated to improving results, and may stop before having enough information to make informed decisions.
- If there is too much tuning, then the data may be split beyond which there is a positive gain; or may be split into too narrowly defined ranges. "*Too high of a value might be specific to a certain sample and allow the model to learn relations which might be highly specific to that sample, resulting in over-fitting*"[48]. "*Decision-tree learners can create over-complex trees that do not generalise the data well. This is called overfitting.*" [50] In the case of over-fitting, the datapoints used for predicting may be exceptions leading to inaccurate predictions. Furthermore, with too much tuning, the model may get too deep, may take too long to run without contributing to information gain, and may be a detriment to predicting results because the model is over-fitted, or has used unnecessary, inappropriate or marginally correlated features. For example, if we set the minimum sample to 1, then the model would overfit and learn from all the data points, including the outliers. [49]
- In Decision Tree, the depth should be representative of the number of features, so that they all have a chance of participating in becoming a decision feature, but not split on arbitrary numeric cutoff. [49]
- Picking the right evaluation metrics that can be meaningful and measurable is very important and can be very tricky[42]. It is critical for a successful machine learning system. It is important to pick the best features to use in the model, to reduce the complexity of the model and to reduce memory consumption. The size of the trees can be adjusted for optimum information gain by tuning the parameter values.

In the case of this project, the evaluation was based on getting precision and recall greater than 0.3.

When I was tuning the parameters manually in my code before task 6, I tried using min_samples_split = 2, 20 and 30 on Tree classifier to see if I could raise the accuracy. The accuracy was 80% with split of 20 or 30 and bounced between 76% and 80% with split of 2.

I tried using various values for Random Seed, Test Percent, Error Percent, Max Deferral Payments, Max Bonus and Split Percent.

I did not do this well because I am a novice, and I did not understand the importance of GridSearch. By not doing it well, I wasted a lot of time hunting and pecking for the right values for the parameters and never getting to the optimal precision and recall scores.

In task 6, when I created a pipeline of the various values to use for Decision Tree, as shown in question 3 above, and used GridSearch to loop through those values to find the best combination, for the best precision and recall results, it went a lot faster. Furthermore, by combining that with SelectKBest, the best features were chosen.

In task 6, I also chose to use GaussianNB even though it is a classifier that does not have any parameters. I ran it through SelectKBest to get the best features. Theoretically, I did not have to run it through GridSearch because there are no parameters, but I ran it through GridSearch anyway because it was simpler to just use the same code that I had used for DecisionTree, including running it through tester.py to get the final results. I was also curious to see how much longer it would take GaussianNB to run through GridSearch with SelectKBest range (1,10). For the 20 features that I had, it did not take that much longer than manually running the classifier.

5. *What is validation, and what's a classic mistake you can make if you do it wrong? How did you validate your analysis? [relevant rubric items: "discuss validation", "validation strategy"]*

Validation is a process to evaluate if the model, that was generated through parameter tuning and training on a subset of the dataset, can be relied upon to predict accurate results. A test dataset is run through the model and the results are evaluated to see if they are sufficiently accurate to use the model to predict outcomes on unseen data. In some situations, a validation dataset is held back as a subset of the training data to get a final objective view of the results generated by the model before using the model to predict outcomes. [46]

I performed validation on my data by having both training and test datasets. I did not hold back a validation dataset as a subset of my training data.

In my code on the features that I selected by hand, I used sklearn cross_validation to split the data into training and testing datasets. And in the second part of my code I used my own definition to split the data using a test % and a split %, following code learned in the course:

```
def complexTrainTestData(data, randomSeed, testPct, errorPct, maxDeferral, maxBonus,
splitPct):
```

Overfitting is a classic mistake if the machine learning model is not done properly and may negatively impact the model's ability to generalize on new data. If the model's parameters are over-tuned, there could be too much granularity in the learned data, resulting in poor results when predicting outcomes on unseen data. [46]

I tried using PCA in my task6, but I did not get good enough performance, so I switched to SelectKBest.

If the dataset has a pattern, then it's necessary to shuffle it to break up the pattern and get representation in training and testing. In my task6, I used StratifiedShuffleSplit with 100 shuffles in my cross-validation, so that the training data would not be overfit on the learned model.

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"]

In the first part of my code where I selected my own two features, I have extensive reports that show some of sklearn's scores:

- My first comparison using sklearn cross_validation, with classifiers GaussianNB, DecisionTree and Random Forest.
- My second comparison using test % and split % for creating the test and training data, with the above 3 classifiers as well as SVM.
- All of the following "Average" parameters for sklearn's precision, recall and F1 metrics, through a loop to get results for each: Binary, Macro, Micro and Weighted.
- All of the following statistics, through a loop to get the results of each: Test Case Size, Length of True Positives, Length of False Positives, Length of True Negatives, Length of False Negatives, Precision, Recall and F1.

The reports are quite extensive, and too long for this document. You may run my code to see the complete reports. Here is the best result from each of "sklearn cross_validation / basic" report and the "complex / split" report:

The best F1 score when running **sklearn cross_validation (basic)** was with **GaussianNB** sklearn metrics' Average Parameter = **micro**. The micro parameter: *"Calculate metrics globally by counting the total true positives, false negatives and false positives".*[47]

Test Case Size	Len True Positive	Len False Positive	Len True Negative	Len False Negative	Precision	Recall	F1
-------	------------	------------	------------	------------	------------	------------	------------
30%	0.000	1.000	22.000	7.000	0.733	0.733	0.733

The best F1 score when running **split (complex)** was with **GaussianNB** with sklearn metrics' Average Parameter = **weighted.** This excludes the results where the precision was 1.0 (binary), because a perfect precision score is not likely: Binary: "*Only report results for the class specified by pos_label. This is applicable only if targets (y_{true,pred}) are binary.*"[47] The weighted parameter: *"Calculate metrics for each label, and find their average, weighted by support (the number of true instances for each label). This alters "macro" to account for label imbalance; it can result in an F-score that is not between precision and recall."*[47]

Avg Parameter	Test Case Size	Len True Positive	Len False Positive	Len True Negative	Len False Negative	Precision	Recall	F1
--------------	-------	------------	------------	------------	------------	------------	------------	------------
Weighted	25%	14.000	0.000	9.000	2.000	0.935	0.920	0.927
Binary (default)	25%	14.000	0.000	9.000	2.000	1.000	0.875	0.933

My identifier has a really great F1 score and very good recall and very good precision scores.

Both my false positive rate (0) and false negative rate (2) are low, which means that I can identify POI's reliably and accurately.

"Precision (P) is defined as the number of true positives (T_p) over the number of true positives plus the number of false positives (F_p)." [53]

"Recall (R) is defined as the number of true positives (T_p) over the number of true positives plus the number of false negatives (F_n)." [53]

The high precision score, with high True Positive Rate and low False Positive Rate, means that whenever a POI gets flagged in my test set, I know with a lot of confidence that it's very likely to be a real POI and not a false alarm. The classifier is returning accurate results.

The high recall score, with High True Positive Rate and low False Negative rate means that I am accurately flagging most of the POIs even though I may have also flagged some non-POIs as POIs. I can put the cuffs on all of them and call them all in for questioning, and in doing so, I am likely not missing the real POIs. The classifier is recalling the majority of all correct results. [54]

"The objective is to get a system with high precision and high recall which will return many results, labelled correctly." [53]

7. Conclusion

I learned a lot doing this project. I worked on it part time for 2 ½ months. I didn't just want to answer the minimum required, because I am very interested in Machine Learning and understanding it. I know I have a long way to go. I am also new at programming in Python, which means that it took me a lot longer to code. At the end of December, I took an objective look at my code and it was awful. It was not organized and it was all over the place. So, I took the extra time to organize it into definitions and loops.

I had tried to select my own features, but they did not pass the minimum precision and recall required for the tester.py program. They appeared to have high precision and recall when I ran them through my own algorithms. However, when they were run through multiple iterations in the tester.py code, they did not meet the minimum. So, I had to go to the forum and to my mentor to find out what to do to improve the scores. This required a whole new approach to selecting features and learning how to tune the algorithms through pipeline and GridSearch. These are valuable skills which I will carry forward into my career. I'm glad that I had to take the extra month to learn them.

In my final code, tasks 1 through 5 represent my initial learning curve features, algorithms and code. I put all of my final approach with SelectKBest to pick the features and pipeline / GridSearch in task 6.

- The end –