Machine Learning Engineer Nanodegree

Capstone Project

Emmanuel Perez
June 26th, 2017

I. Definition

Project Overview

Product categorization is a problem that affects many companies around the world, companies like Amazon, Walmart get thousands of products every day and they need to quickly categorize all these products in a manner so they can get those products to the right users. This is not an easy task to do, but now thanks to the technology advances in Machine Learning we can find new and exciting ways to solve this type of problems, one of them is Supervised Machine Learning using Classification algorithms.

Supervised machine learning is the search for algorithms that reason from externally supplied instances to produce general hypotheses, which then make predictions about future instances. In other words, the goal of supervised learning is to build a concise model of the distribution of class labels in terms of predictor features. The resulting classifier is then used to assign class labels to the testing instances where the values of the predictor features are known, but the value of the class label is unknown.

(https://datajobs.com/data-science-repo/Supervised-Learning-[SB-Kotsiantis].pdf)

In this Capstone Project, I had the option to select a topic in Kaggle, so found a competition that I think suits my needs for this project, is the "Otto Group Product Classification Challenge", The Otto Group is one of the world's biggest e-commerce companies, with subsidiaries in more than 20 countries, including Crate & Barrel (USA), Otto.de (Germany) and 3 Suisses (France). For this competition, they provide a dataset with 93 features for more than 200,000 products. The objective is to build a predictive model which can distinguish between the main product categories.

(Kaggle, https://www.kaggle.com/c/otto-group-product-classification-challenge)

Problem Statement

A consistent analysis of the performance of Otto Group's products is crucial. However, due to their diverse global infrastructure, many identical products get classified differently. Therefore, the quality of their product analysis depends heavily on the ability to accurately cluster similar products. The better the classification, the more insights they can generate about our product range.

We can improve the classification of products by using Supervised Learning algorithms and train a model to predict to which Category a product belongs to, the total number of categories are nine for all products.

The strategy to solve this problem will be to train three different Classification learners which are Naïve Bayes Classifier, Gradient Boosting Classifier and Random Forest Classifier, after that we will be comparing their results and for the last step we are going to select the one with better results in the f1_score to test and give the las results.

Metrics

I will be using the F1 Score obtained where the score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0.

Formula:

```
F1 = 2 * (precision * recall) / (precision + recall)
```

For this specific problem of classification we have a multi class case, where we need to use a weighted average of the F1 score of each class.

The precision is the ratio `tp / (tp + fp)` where `tp` is the number of true positives and `fp` the number of false positives. The precision is intuitively the ability of the classifier not to label as positive a sample that is negative.

The recall is the ratio `tp / (tp + fn)` where `tp` is the number of true positives and `fn` the number of false negatives. The recall is intuitively the ability of the classifier to find all the positive samples.

(http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html)

II. Analysis

Data Exploration

Each row of the data corresponds to a single product. There is a total of 93 numerical features, which represent counts of different events. All features have been obfuscated and will not be defined any further.

There are nine categories for all products. Each target category represents one of our most important product categories (like fashion, electronics, etc.). The products for the training and testing sets are selected randomly.

The training Dataset has a total of 61,878 products with a total of 93 numerical features and 0 categorical features, one "id" column and a target column that refers to the Class/Category of the product. The products are divided in the following proportion:

Class_1	Class_2	Class_3	Class_4	Class_5	Class_6	Class_7	Class_8	Class_9
1929	16122	8004	2691	2739	14135	2839	8464	4955

Statistics for the first 10 features:

	feat_1	feat_2	feat_3	feat_4	feat_5	feat_6	feat_7	feat_8	feat_9	feat_10
count	61878.00000	61878.000000	61878.000000	61878.000000	61878.000000	61878.000000	61878.000000	61878.000000	61878.000000	61878.000000
mean	0.38668	0.263066	0.901467	0.779081	0.071043	0.025696	0.193704	0.662433	1.011296	0.263906
std	1.52533	1.252073	2.934818	2.788005	0.438902	0.215333	1.030102	2.255770	3.474822	1.083340
min	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000
max	61.00000	51.000000	64.000000	70.000000	19.000000	10.000000	38.000000	76.000000	43.000000	30.000000

Statistics for the last 10 features:

feat_84	feat_85	feat_86	feat_87	feat_88	feat_89	feat_90	feat_91	feat_92	feat_93
61878.000000	61878.000000	61878.000000	61878.000000	61878.000000	61878.000000	61878.000000	61878.000000	61878.000000	61878.000000
0.070752	0.532306	1.128576	0.393549	0.874915	0.457772	0.812421	0.264941	0.380119	0.126135
1.151460	1.900438	2.681554	1.575455	2.115466	1.527385	4.597804	2.045646	0.982385	1.201720
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000
76.000000	55.000000	65.000000	67.000000	30.000000	61.000000	130.000000	52.000000	19.000000	87.000000

Exploratory Visualization

I ran a PCA analysis of the features and got to the conclusion that the features in this dataset are not that much correlated, after a few try's I got the explained variance sum to 1 with a number of components of 80, this indicates that the number of features reduced is from 93 to 80, the difference is not that big, that's why I'm not included a PCA feature reduction as a part of the training, in the repository you can find a chart showing this behavior, the image is called Figure PCA. After running the commands

```
pca_results = vs.pca_results(training_data, pca)

print pca_results['Explained Variance'].cumsum()
```

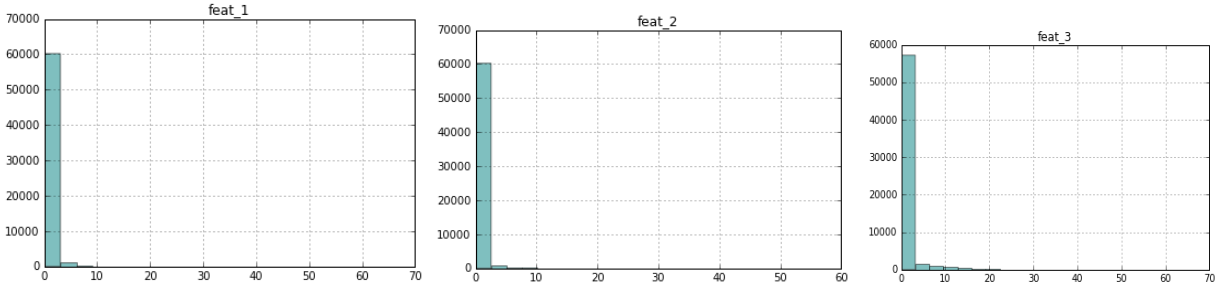
I got the following explained variance by dimension:

Dimension 1	0.1177	Dimension 51	0.9352
Dimension 2	0.2038	Dimension 52	0.9383
Dimension 3	0.2677	Dimension 53	0.9414
Dimension 4	0.3290	Dimension 54	0.9444
Dimension 5	0.3731	Dimension 55	0.9472
Dimension 6	0.4128	Dimension 56	0.9499
Dimension 7	0.4477	Dimension 57	0.9525
Dimension 8	0.4792	Dimension 58	0.9550
Dimension 9	0.5086	Dimension 59	0.9574
Dimension 10	0.5361	Dimension 60	0.9598
Dimension 11	0.5620	Dimension 61	0.9621
Dimension 12	0.5843	Dimension 62	0.9644
Dimension 13	0.6059	Dimension 63	0.9666
Dimension 14	0.6256	Dimension 64	0.9687
Dimension 15	0.6448	Dimension 65	0.9707
Dimension 16	0.6634	Dimension 66	0.9726
Dimension 17	0.6809	Dimension 67	0.9744
Dimension 18	0.6980	Dimension 68	0.9761
Dimension 19	0.7135	Dimension 69	0.9778
Dimension 20	0.7276	Dimension 70	0.9794
Dimension 21	0.7400	Dimension 71	0.9809

Dimension 22 0.7518 Dimension 23 0.7625 Dimension 24 0.7727 Dimension 25 0.7827 Dimension 26 0.7920 Dimension 27 0.8006 Dimension 28 0.8089 Dimension 29 0.8170 Dimension 30 0.8245 …	Dimension 72 0.9823 Dimension 73 0.9837 Dimension 74 0.9850 Dimension 75 0.9863 Dimension 76 0.9875 Dimension 77 0.9887 Dimension 78 0.9898 Dimension 79 0.9908 Dimension 80 0.9918

As we can see 99.18% of the explained variance is reached in the 80th component.

Also did some prints of every feature, I included in the report 3 figures representing the first 3 features and their histogram, we did this to observe the distribution for every one of the features and I came to the conclusion that each of the features follow a similar distribution, skewed.



To download the data, you need to have an account in Kaggle and accept the rules of the challenge, the dataset size is 12mb aprox.

https://www.kaggle.com/c/otto-group-product-classification-challenge/rules

Algorithms and Techniques

The selected algorithms were:

1) Naïve Bayes

- In machine learning, naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features.
    - https://en.wikipedia.org/wiki/Naive_Bayes_classifier

- I decided to implement a simple Naïve Bayes classifier and focus on the mentioned features. The simplicity of Naïve Bayes makes for a great testing environment and feature building.

2) Boosting

- Boosting is a machine learning ensemble meta-algorithm for primarily reducing bias, and also variance in supervised learning, and a family of machine learning algorithms which convert weak learners to strong ones.
    - https://en.wikipedia.org/wiki/Boosting_(machine_learning)
- I selected Boosting algorithms because is one of the most widely used algorithm in data science competitions, and also according to a research I made they're used a lot in problems of text categorization problems so I though it can be useful for this problem.

3) Random Forest (For the benchmark)

- Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.
    - https://en.wikipedia.org/wiki/Random_forest
- The purpose of Random Forest is to do a benchmark against the selected algorithm.

4) GridSearch

- To help us systematically working through multiple combinations of parameter tunes, cross validate each and determine which one gives the best performance. You can work through many combinations only changing parameters a bit.
    - https://stackoverflow.com/questions/19335165/cross-validation-and-grid-search

Benchmark Model

As benchmark, I will be using a Random Forest model, I'm planning to compare the Random Forest F1 Score to the F1 Score obtained for the Capstone Project, we will be training/testing this model with the same process of the other classifiers following the Project Design described below.

III. Methodology

Data Preprocessing

I did not have the need to preprocess the provided data, As I explained before all the features were numerical values, so no need to apply some transformation to categorical features, also I checked the histograms for all features and all followed a similar distribution, and the PCA process reveal that there was no need to reduce the feature space because the explained variance was accumulated in the first 80 principal components.

I used the training train_test_split method of sklearn.model_selection to divide the data into Training/Test datasets, using a percentage of 75% (46,408 samples) for training and 25%(15,470 samples) for testing, also used a random state of 42.

Implementation

First, I needed to implement a model evaluation code, for this I used the utility code provided by Udacity for the first project of "Student Intervention":

- train_classifier(clf, X_train, y_train): This function receives the features/target data and the selected classifier, then use this parameters to fit the data with the model and then print the time in seconds of the model.

- predict_labels(clf, features, target): this function receives the selected classifier, the features and the target values, then this parameters are used to make predictions using the trained classifier and then use the f1_score function of sklearn to calculate the performance of the predictions against the target values.

- train_predict(clf, X_train, y_train, X_test, y_test):This function receives the classifier and the training and testing data, then proceeds to call the

train_classifier function and then use the predict_labels function to print the f1_score.

The models used for the experiment were Gaussian Naïve Bayes(GaussianNB), Gradient Boosting Classifier(GradientBoostingClassifier) and Random Forest Classifier(RandomForestClassifier), then I used the train_predict function to run the experiments with the three different models. After the models were trained we got the f1_score per every model and the time we can see those results in the following tables:

Classifer 1 - Gaussian Naive Bayes?

Training Set Size	Training Time	Prediction Time (test)	F1 Score (train)	F1 Score (test)
46408	0.1500 seconds	0.4550 seconds	0.6210.	0.6272.

Classifer 2 - Gradient Boosting Classifier?

Training Set Size	Training Time	Prediction Time (test)	F1 Score (train)	F1 Score (test)
46408	120.0350 seconds	1.3220 seconds.	0.7942.	0.7772.

Classifer 3 - Random Forest?

Training Set Size	Training Time	Prediction Time (test)	F1 Score (train)	F1 Score (test)
46408	1.1260 seconds	0.0590 seconds.	0.9927.	0.7766.

Choosing the Best Model

In the tables above we can now see the different results, for the first classifier GNB, the result was not that good, F1 Score is in 0.6210 for training and for testing is 0.6272 with a training time of 0.1500 seconds, I believe that this result is like this because of the distribution that the data follows.

In the second classifier GBC, we can see better results compared to the first classifier but worst in terms of training time, I believe that the training takes longer because of the nature of Gradient Boosting being a serial process, it takes a total of 120 seconds to train even though the testing is a lot faster, the reported f1_score for training was 0.7942 and 0.7772 for testing definitely an improvement.

For the last case RFC, we have that the training time is faster and the prediction time also is fast, the f1_score for training is incredible with 0.9927 and the f1 score for testing is like the one in the second classifier with 0.7766.

Complications

I think that the only complication really was to select the right parameters to tune the Gradient Boosting algorithm with Grid Search, in some cases the obtained results were not improved and in other cases my machine just stay training for an infinite time consuming all the resources of my laptop, but after some research I got to the right parameters as we can see in the next section.

Selected Model

Based on the experiments performed, I decided to select Gradient Boosting Classifier as the model to be tuned, I believe that Gradient Boosting is one of the best classifiers and think that fits the solution for this problem as well.

Refinement

After selecting the model GBC, the next task is try to tune the model to get better results, after researching more about GBC I found that we can tune a lot of parameters, I did some testing tuning some of this parameter and used the following parameter configuration:

- learning_rate=0.1
- min_samples_split=500
- min_samples_leaf=50
- max_depth=8
- max_features='sqrt'
- subsample=0.8
- random_state=42.

I used GridSearchCV from sklearn to tune the GBC model,with the above parameters, and after the execution of the tuned classifier I got that the f1_score for training is 0.8487 and for testing was 0.8023, definitely an improvement over the previous implementation which had a f1_score for training of 0.7942 and for testing of 0.7772.

(https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/)

IV. Results

Model evaluation and validation

Like other boosting methods, gradient boosting combines weak "learners" into a single strong learner, in an iterative fashion. We selected in my opinion one of the best model to handle classification problems, I believe that the final parameters used for the last experiment were great to improve the f1 score, I am also satisfied with the final results of GBC and believe that is robust enough at least for this problem.

I believe that the selected model is robust enough to solve this problem, it shown a good performance regarding the F1 Score and a considerable improvement after tuning the parameters for Gradient Boosting.

I also believe that even though the results were good for this data we can't really assure that this model will be a good in every case, for that I think we need to know a little bit more about the organization and exactly what the features represent, but we can say that for this problem the selected Model is a good fit.

In conclusion we can say that the selected model results have a reasonable performance and align with the expectations we had at first to solve this problem.

Justification

As a benchmark, we are going to use the result from a tuned Random Forest Classifier, we execute this experiment using the same process as we did for the Gradient Boosting Classifier, but tuning parameters related to the model.

F1 Score Gradient Boosting Classifier		F1 Score Random Forest	
Training	Testing	Training	Testing
0.8487	0.8023	1.0000	0.7999

As we can see in the table, the training f1 score of GBC is 0.8487 and the training f1 score of the benchmark is 1, this tells us that the RFC is better to classify seen data, but the GBC

is slightly better when it comes to classify unseen data, as we can see with the training f1 score of RFC that is 0.7999 lower than 0.8023.

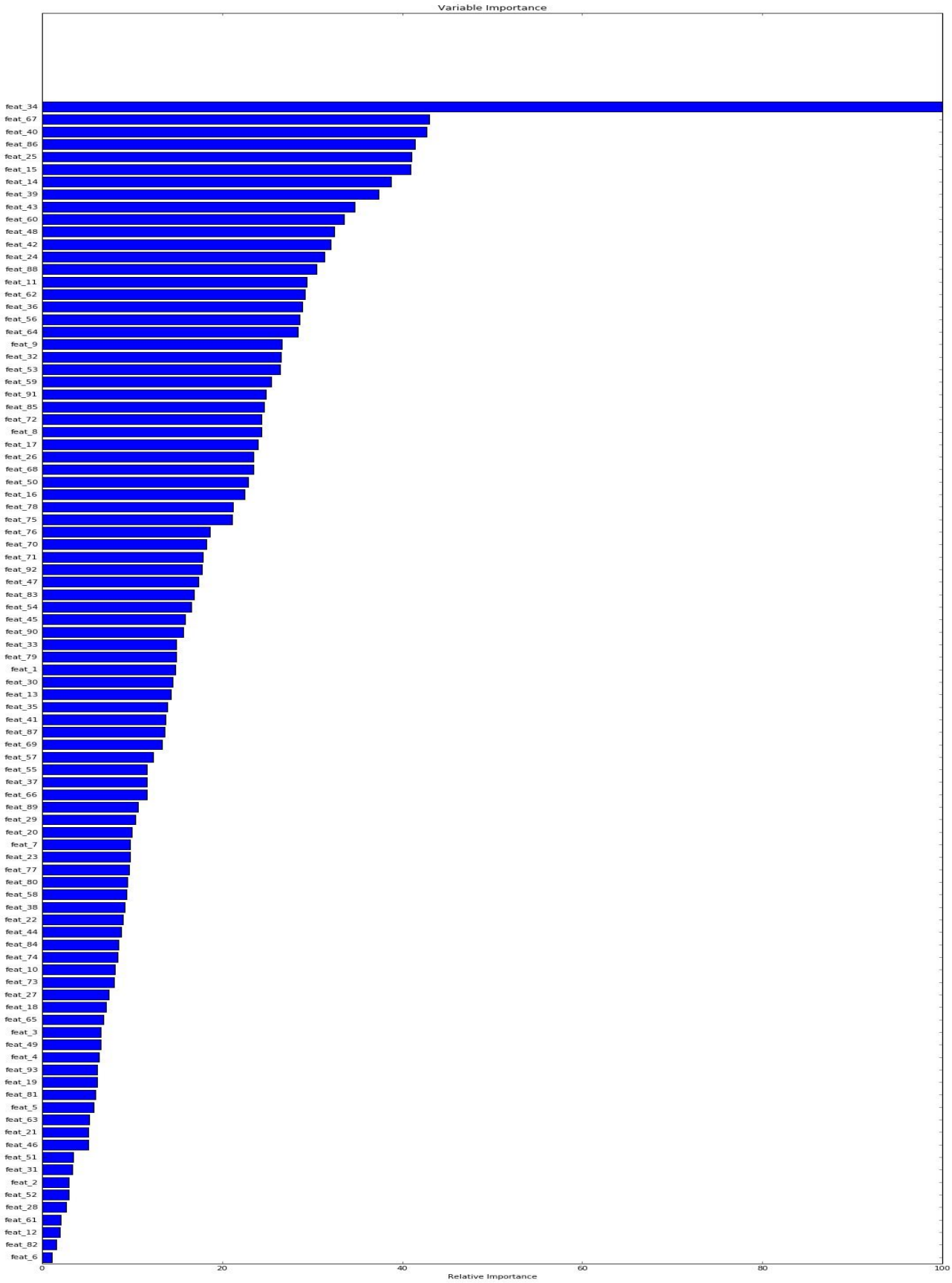
After this result, I can only assume that the Gradient Boosting Classifier has no major difference in predictions compared to Random Forest, so I think that any of this models will do the job because they are very similar when it comes to performance.

V. Conclusion

Free-Form Visualization

Something about the project that caught my attention was the features importance, in a previous analysis I concluded that we could not do a very significant impact in the results by reducing the feature space, in the following chart we can see that behavior, I plotted the Features Importance for the Boosting Classifier:

From a Scale of 1 to 100 we defined the Relative Importance, and for every feature we can see that the first feature (feat_34) have a bigger proportional importance compared to the other features, but after that we can see a “uniform” behavior from the other features forward, and this is like the analysis I ran with PCA, the majority of features in my opinion have a similar importance for the final results.


Variable Importance
feat_34
feat_67
feat_40
feat_86
feat_25
feat_15
feat_14
feat_39
feat_43
feat_60
feat_48
feat_42
feat_24
feat_88
feat_11
feat_62
feat_36
feat_56
feat_64
feat_9
feat_32
feat_53
feat_59
feat_91
feat_85
feat_72
feat_8
feat_17
feat_26
feat_68
feat_50
feat_16
feat_78
feat_75
feat_76
feat_70
feat_71
feat_92
feat_47
feat_83
feat_54
feat_45
feat_90
feat_33
feat_79
feat_1
feat_30
feat_13
feat_35
feat_41
feat_87
feat_69
feat_57
feat_55
feat_37
feat_66
feat_89
feat_29
feat_20
feat_7
feat_23
feat_77
feat_80
feat_58
feat_38
feat_22
feat_44
feat_84
feat_74
feat_10
feat_73
feat_27
feat_18
feat_65
feat_3
feat_49
feat_4
feat_93
feat_19
feat_81
feat_5
feat_63
feat_21
feat_46
feat_51
feat_31
feat_2
feat_52
feat_28
feat_61
feat_12
feat_82
feat_6
0
20
40
60
80
100
Relative Importance

Reflection

1) The first step was to analyze the data for the problem, first I evaluated the categorical and numerical features of the data and also the target column to see which classes/categories where present in the data, we did not have to do major changes I believe that the Otto company preprocess the data before the competition started. We tried to reduce the dimensionality of the data but I came to the conclusion that this step was not necessary given the nature of the data, we also analyzed the type of distribution per every feature.
2) The next step was to separate the data in training/testing with a proportion of 75%/25%, this to help us train a model and then test It using the test data with the target column removed.
3) Then we tested three different model Gaussian Naïve Bayes, Gradient Boosting Classifier and Random Forest Classifier, we ran some experiments using the default parameters values, after this experiment we selected the best model for the problem based on his f1 score.
4) Then we tuned the selected model using GridSearchCV, we defined a f1_scorer function using the make_scorer function of sklearn, and then used this to build the GridSearchCV execution, using the selected parameters we tuned the classifier and then got the improved results.
5) After the improvement, we compared the Selected model with the Benchmark model and got to the conclusion that both models have a similar behavior regarding this problem, so I think we can use anyone of them, just the GBC is a little bit better regarding the f1 score.

One of the things that I found interesting for this project was the distribution that follows each of the features, was like a skewed distribution, I thought that this was going to be a problem, but later in the implementation I found that the data was just right to be used as training data.

Another thing that I believe is interesting is that the model I selected as a benchmark resulted to have a better f1 score than the other models at least for the training, without a doubt the RFC had better performance for training but is very similar to the f1 score for the testing.

Improvement

I think there is always room for improvement in machine learning, in the case of this project I would say that maybe researching a little bit more, we can find some parameters that will improve for sure our results.

Another thing that can improve the results I think would be to understand more the data, maybe if we could get a more significant description of the features would be better, so with this information we can maybe discard some of the features and leave only the ones we are interested.

Maybe using another of the ensemble methods and compare them with the obtained results we could get some improvement.