Report - Otto Group product classification

Introduction

Client

Data and its acquisition

Data Exploration

- Cleaning
- Is there class imbalance?
- Visualizing the data
- Are some features related to each other?
- Are the correlations statistically significant?
- Important features to predict the target product category
- Features dimension reduction
    - PCA - Principal Component Analysis
    - T-SNE (T-Distributed Stochastic Neighbouring Entities) visualiztion
    - LDA - Linear Discriminant Analysis

Data Modeling

- Theory and Approach followed
- Overfitting and Underfitting
- Model evaluation criteria
- Data and code organization
- First (Baseline) Model
    - Baseline Results
    - Held out test set
    - Logistic Regression

- Dealing with class imbalance
    - Over sampling
    - Under sampling
    - Combination of oversampling and undersampling
    - Balancing using weights

- Other simple Models
    - MultinomialNB
    - Support Vector Machines
    - KNN
- Ensembling and Stacking Models
    - Bagging Models - Random Forest
    - Boosting Models - Gradient Boosting
    - Boosting Models - XGB
- Stacking Models

Model Comparisons

Model Recommendations

Conclusions and Future Analysis

Introduction

Imagine that you are one of the biggest e-commerce companies in the world, with subsidiaries in many countries. Millions of products are sold and thousands are added to the product line every day. A consistent analysis of the products become crucial. But due to diverse global infrastructure, many identical products get classified differently. Therefore the quality of the product analysis depends heavily on the ability to accurately cluster similar products. The better the classification the more insights can be generated about the product range.

Client

Here the client is the Otto-group company, one among the top e-commerce companies. We hope to provide the client with the best of the algorithms that would classify the given products as accurately as possible.

Data and its acquisition

The data is provided by the client in the form of csv file. There is some additional test data that can be used to evaluate and report the algorithm success. The main data file has around 61 thousand products. Each of the product has 93 associated features and the correct class (product category) out of the total 9 classes. The actual meaning of the features is not available. So we can treat them just as numbers. Similarly the 9 classes are just numbers without any meaningful name. The actual meaning of the features and the product categories might have helped us understand the problem better. But this might be the case is many other projects where there are some confidentiality restrictions.

Loading the data to use for modeling is a very simple step. I am using the read_csv function provided by the python-pandas library. It directly loads the data into a pandas dataframe (table like structure) object and can be used for processing readily.

Data exploration

Cleaning

The data provided doesn't have null values or any missing values. The product id field is of no significance and is removed. All the 93 features together identifies the unique product for our purposes and is sufficient. The product classes are in text form and needs to be converted to the numerical values, so as to be consumed by machine learning libraries. We simply number them from 1 to 9.

So our data has 61 thousand rows with 93 feature columns named feat_1, feat_2 ...feat_93 and 1 target column having values from 1 to 9. Here is how it looks:

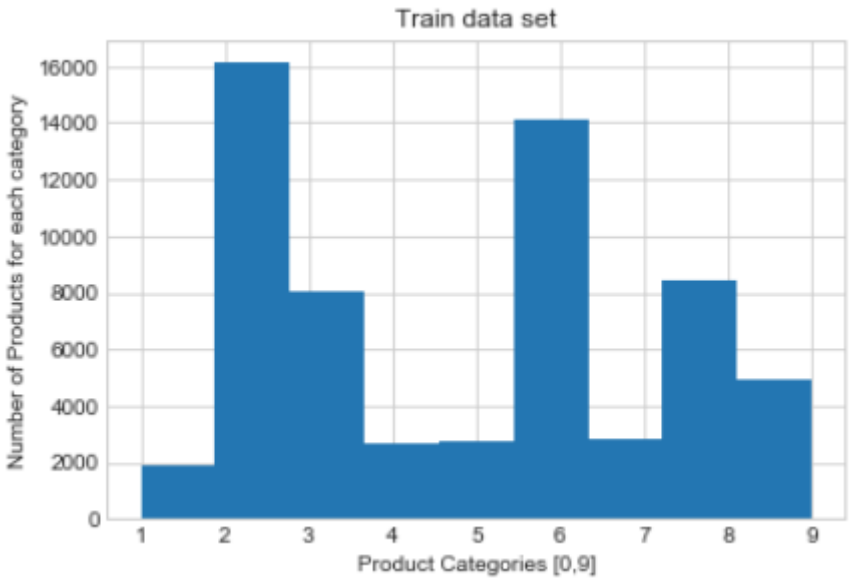
feat_1	feat_2	feat_3	feat_4	feat_5	feat_6	feat_7	feat_8	feat_9	feat_10	...	feat_85	feat_86	feat_87	feat_88	feat_89	feat_90	feat_91	feat_92	feat_93	target
1	0	0	0	0	0	0	0	0	0	...	1	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	...	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	...	0	0	0	0	0	0	0	0	0	1
1	0	0	1	6	1	5	0	0	1	...	0	1	2	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	...	1	0	0	0	0	1	0	0	0	1

Is there any class imbalance?

If there are approximately equal number of rows for each target class we call it "balanced classes" and "imbalanced classes" otherwise. We need to take different approaches in the both the cases. In general, the classes with more data will carry more weightage in our model unnecessarily, while the classes with less data would have negligible effect on our model. This is not what we want. We want the model to have equal representation from all the classes to do accurate predictions. Although imbalanced classes are not always a problem. Various algorithms deal with them internally. It also depends on how much the minority classes are important. Sometimes they are extremely important like in fraud detection cases. In such cases anomaly detection framework can also be used. In general there are few ways to deal with the imbalances classes:

- While in the training phase use stratified sampling
- Oversampling the minority classes
- Undersampling the majority classes
- At algorithm level, adjust the class weights or change the decision threshold
- Don't use accuracy for the model evaluation, use AUC or F1-score instead.

For the problem we are solving here, none of the class is more important than others. Also the class imbalance in the data might be natural i.e. there might be overall more products for certain classes. So we would got with stratified sampling in our model phase. Here is how the classes distribution look like:



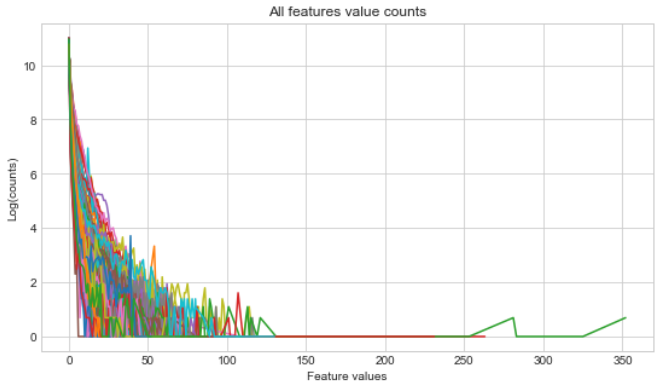
The classes 2 and 6 have relatively more data.

Visualizing the data

Since the data is huge simple reading the data doesn't help. Following visualizations will help understand the data better.

Fig 1. Feature value counts

- 93 features presented as different colors
- The sharp jerky lines show that the feature values are integers
- Most feature values are less than 80
- Most feature values are conentrated in 0-5, hence log of value counts on y axis



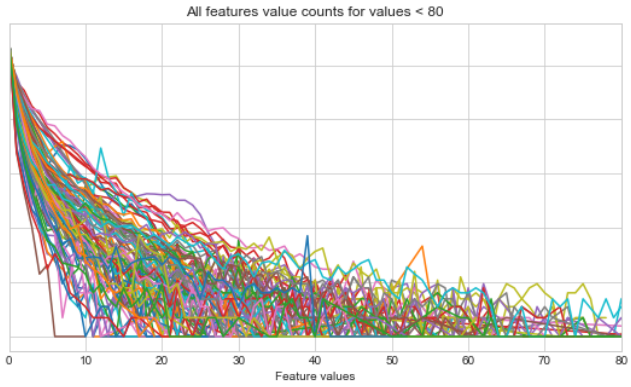
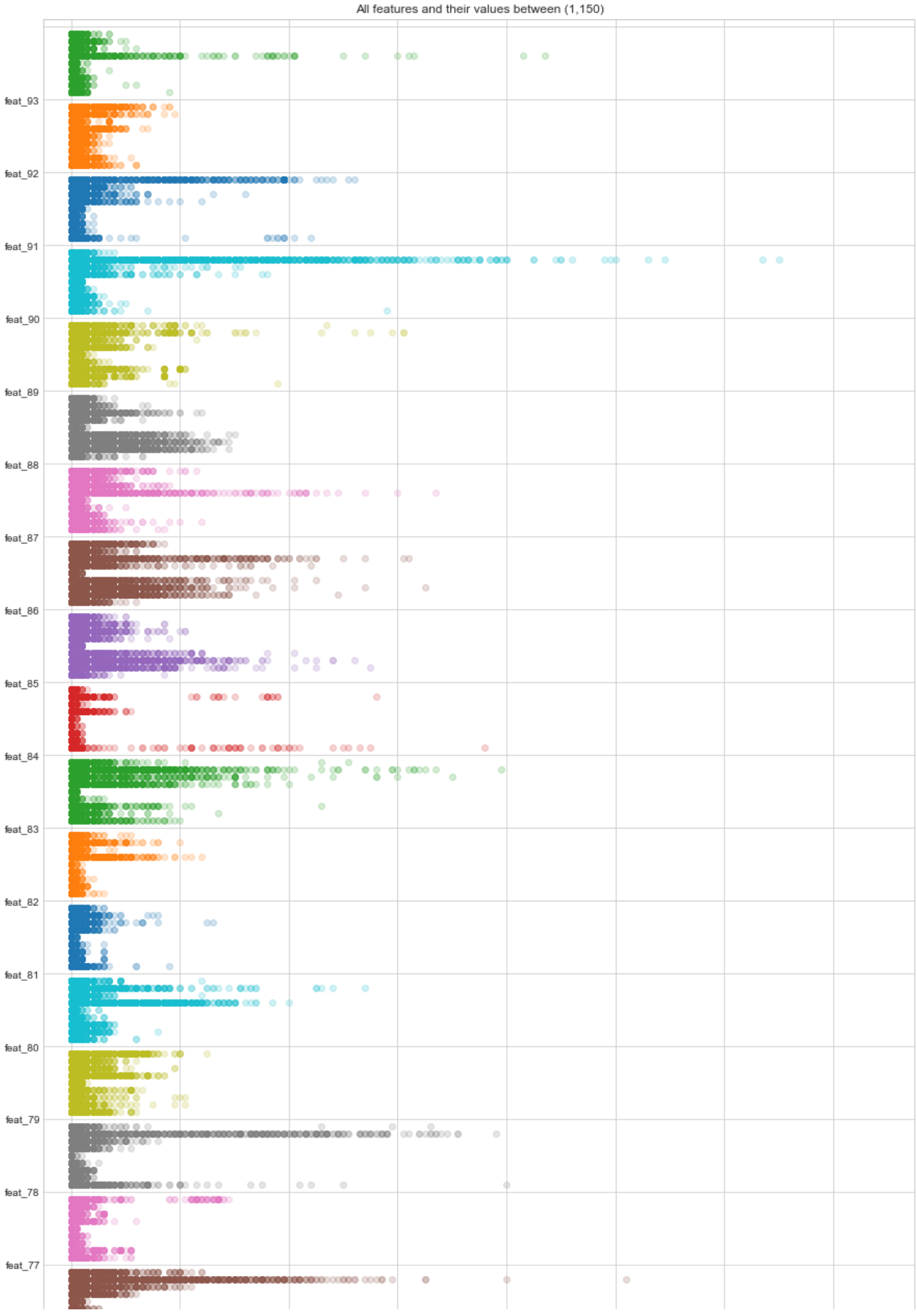
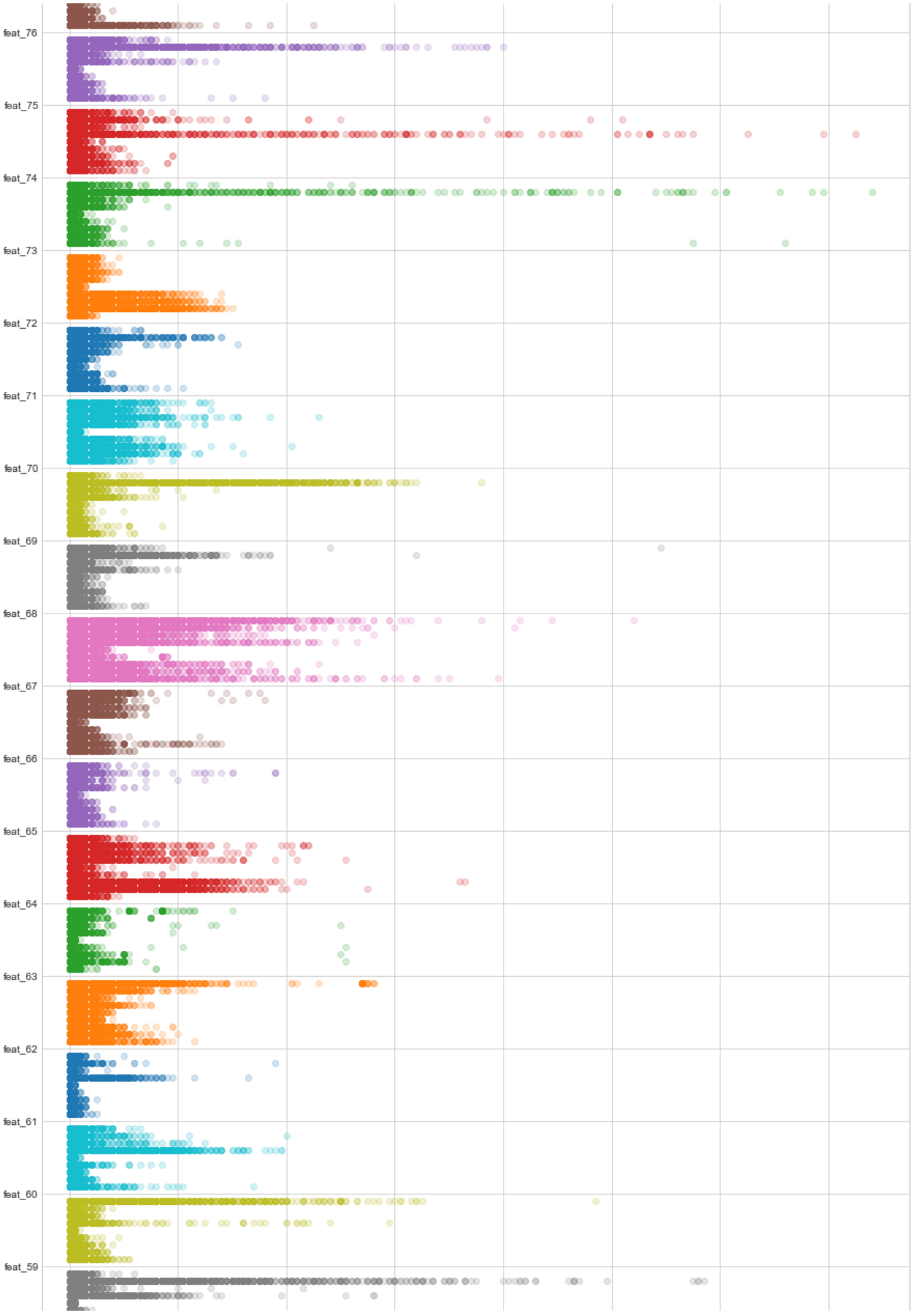


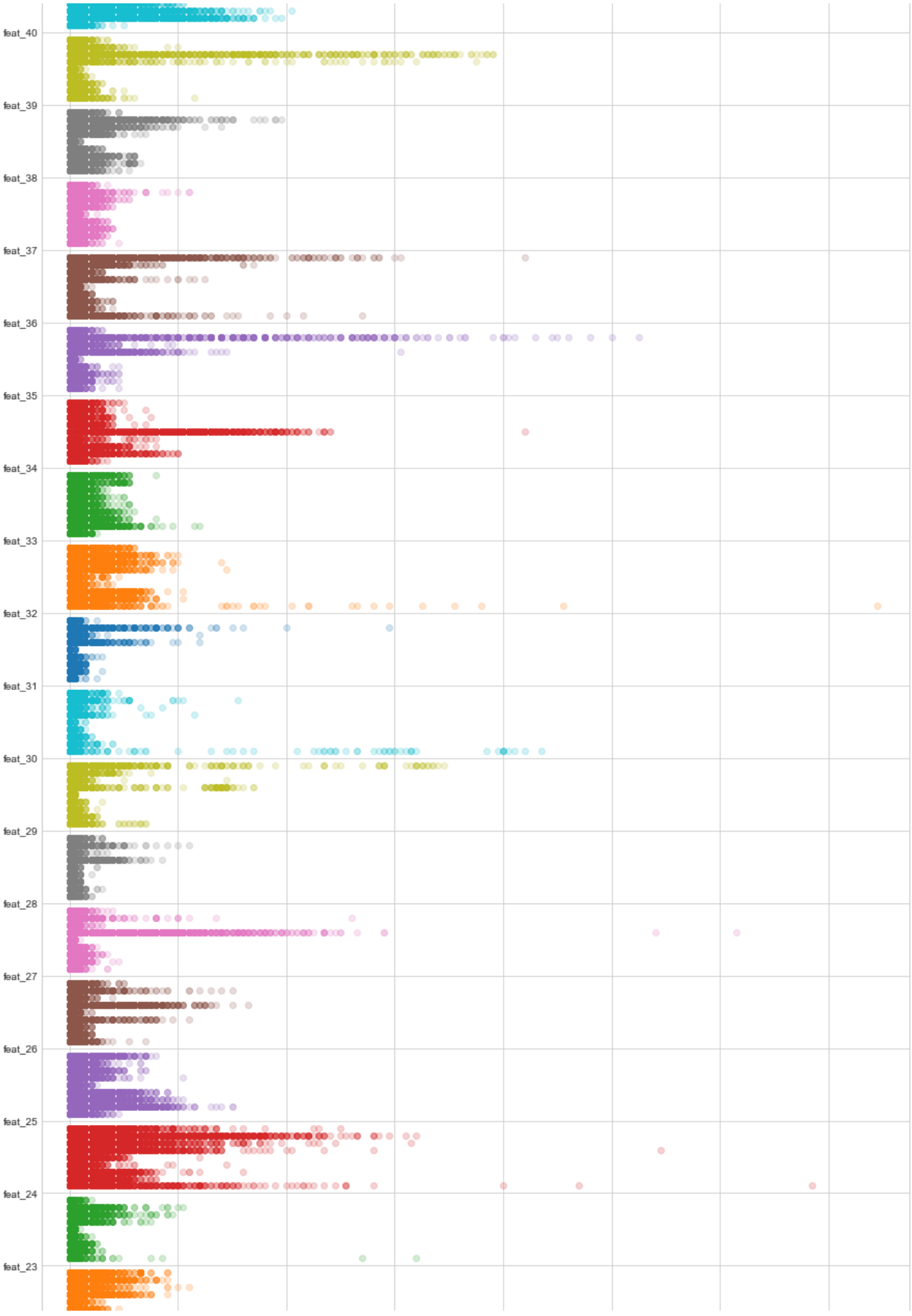
Fig 2. Feature values and their target classes

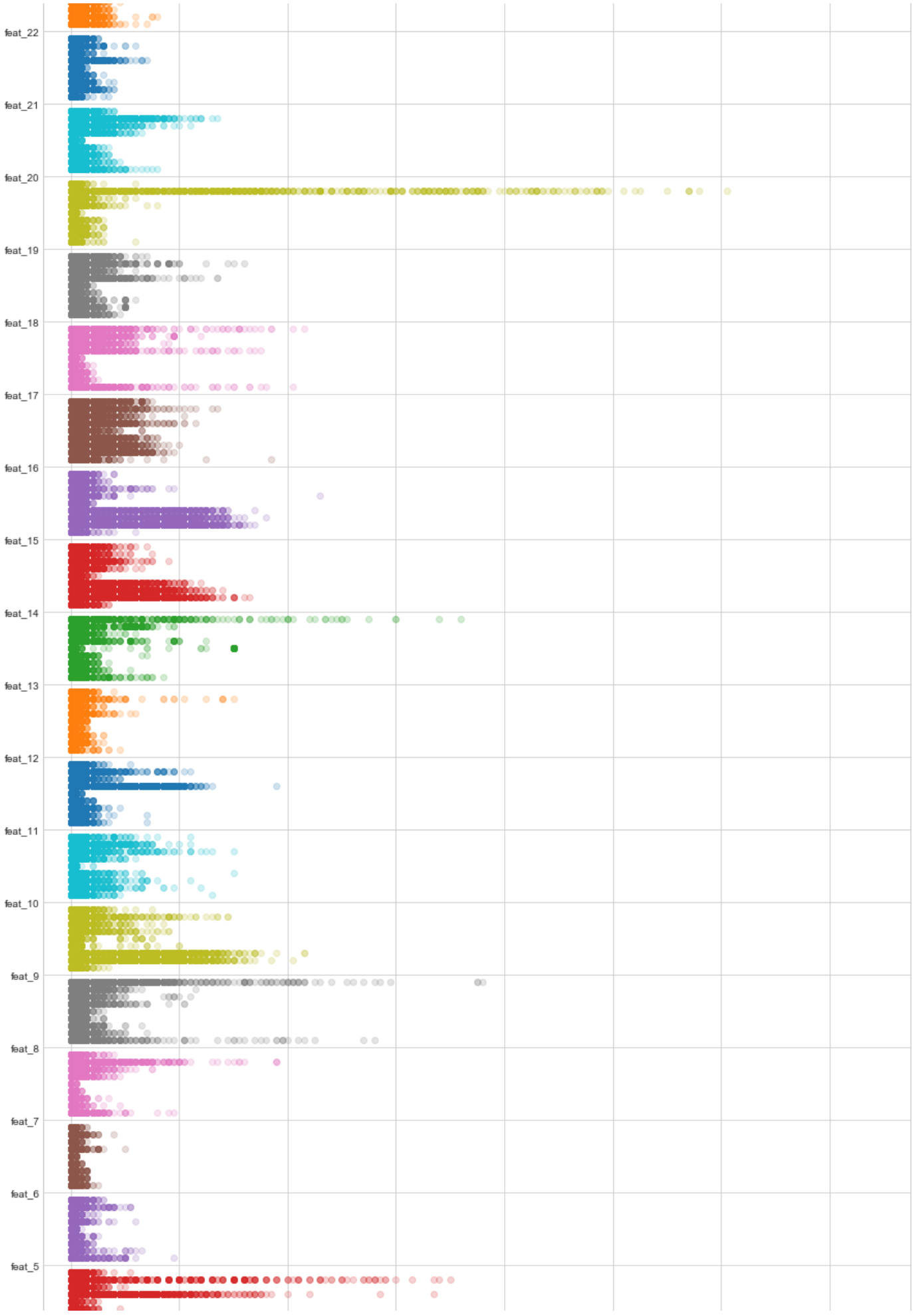
- Shows the range of the values each feature can take
- Each feature has 9 separate rows for each target class
- Two features with extreme values shown separately
- the itensity of colors shows the concentration for that feature value


All features and their values between (1,150)
feat_93
feat_92
feat_91
feat_90
feat_89
feat_88
feat_87
feat_86
feat_85
feat_84
feat_83
feat_82
feat_81
feat_80
feat_79
feat_78
feat_77


feat_76
feat_75
feat_74
feat_73
feat_72
feat_71
feat_70
feat_69
feat_68
feat_67
feat_66
feat_65
feat_64
feat_63
feat_62
feat_61
feat_60
feat_59


feat_58
feat_57
feat_56
feat_55
feat_54
feat_53
feat_52
feat_51
feat_50
feat_49
feat_48
feat_47
feat_46
feat_45
feat_44
feat_43
feat_42
feat_41
Product category 1-9 for all features 1-93


feat_40
feat_39
feat_38
feat_37
feat_36
feat_35
feat_34
feat_33
feat_32
feat_31
feat_30
feat_29
feat_28
feat_27
feat_26
feat_25
feat_24
feat_23


feat_22
feat_21
feat_20
feat_19
feat_18
feat_17
feat_16
feat_15
feat_14
feat_13
feat_12
feat_11
feat_10
feat_9
feat_8
feat_7
feat_6
feat_5

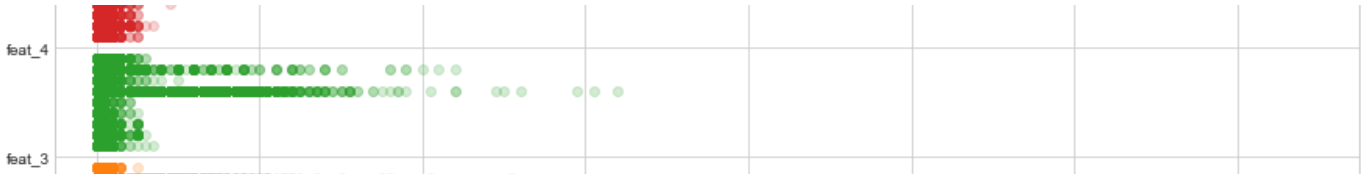
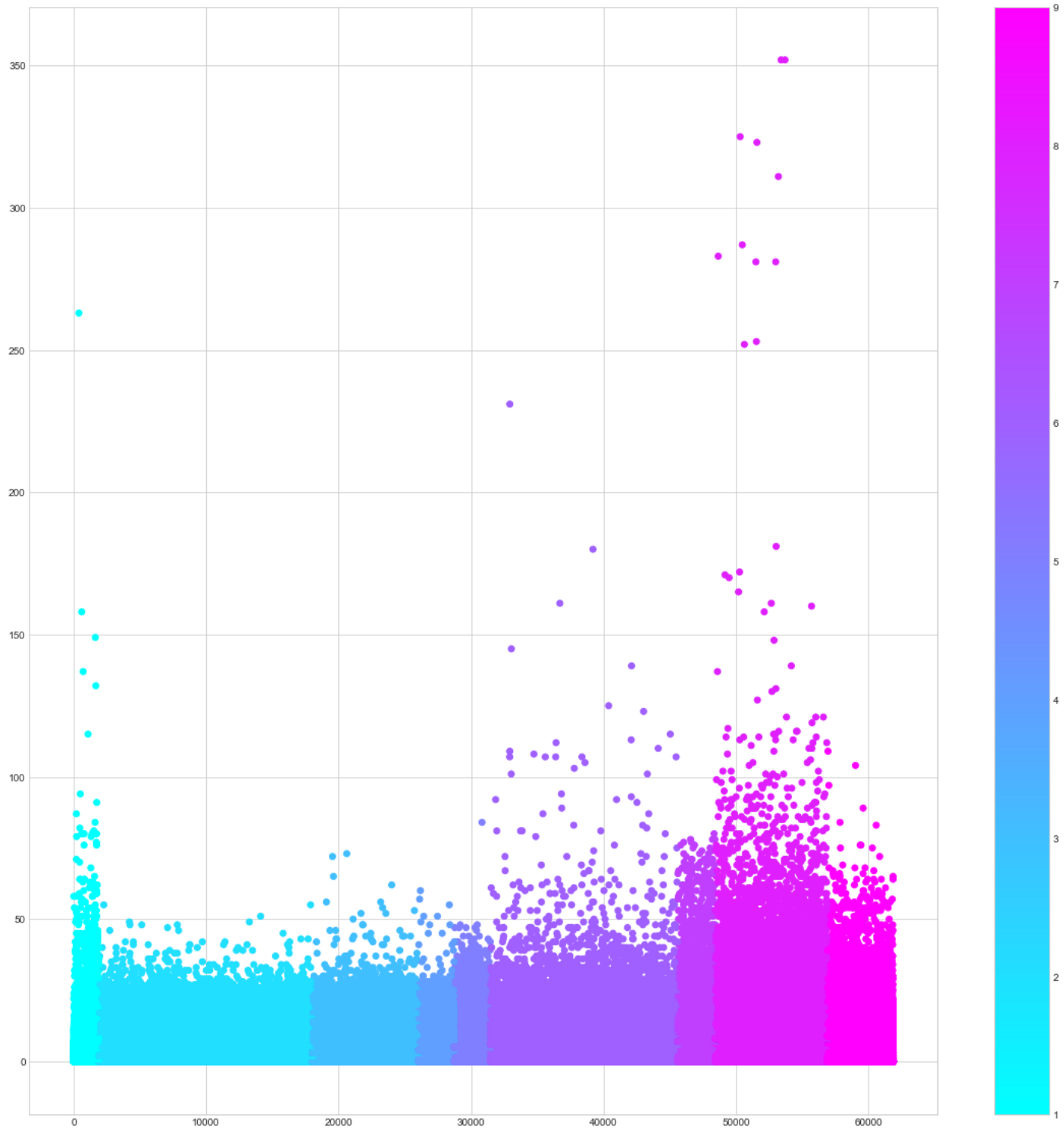

feat_4
feat_3

Fig 3. All 60K feature rows and their target classes

- The width of the colored bars shows the class imbalances
- Shows classes that have extreme feature data values
- Features data overlap on the chart

should put title etc.



Are some features related to each other?

Knowing if some features are related to each other helps. If 2 features are linearly correlated one of them can be removed to produce a simpler ML model that defines relation between features and target.

To find the correlation between 2 features we can find the pearson correlation index for them. If its close to 1 or -1, it would indicate strong linear relationship. But a value near 0 means no linear relation between the 2 features. We have 93 features and we need to find the correlation between all the pairs.

Fig 1. Correlation Matrix

- Features matrix
- Colors bar represents the correlation index variance
- Darker shaded squares have high correlation
- Most high correlations are positive meaning the 2 feature values increase or decrease together

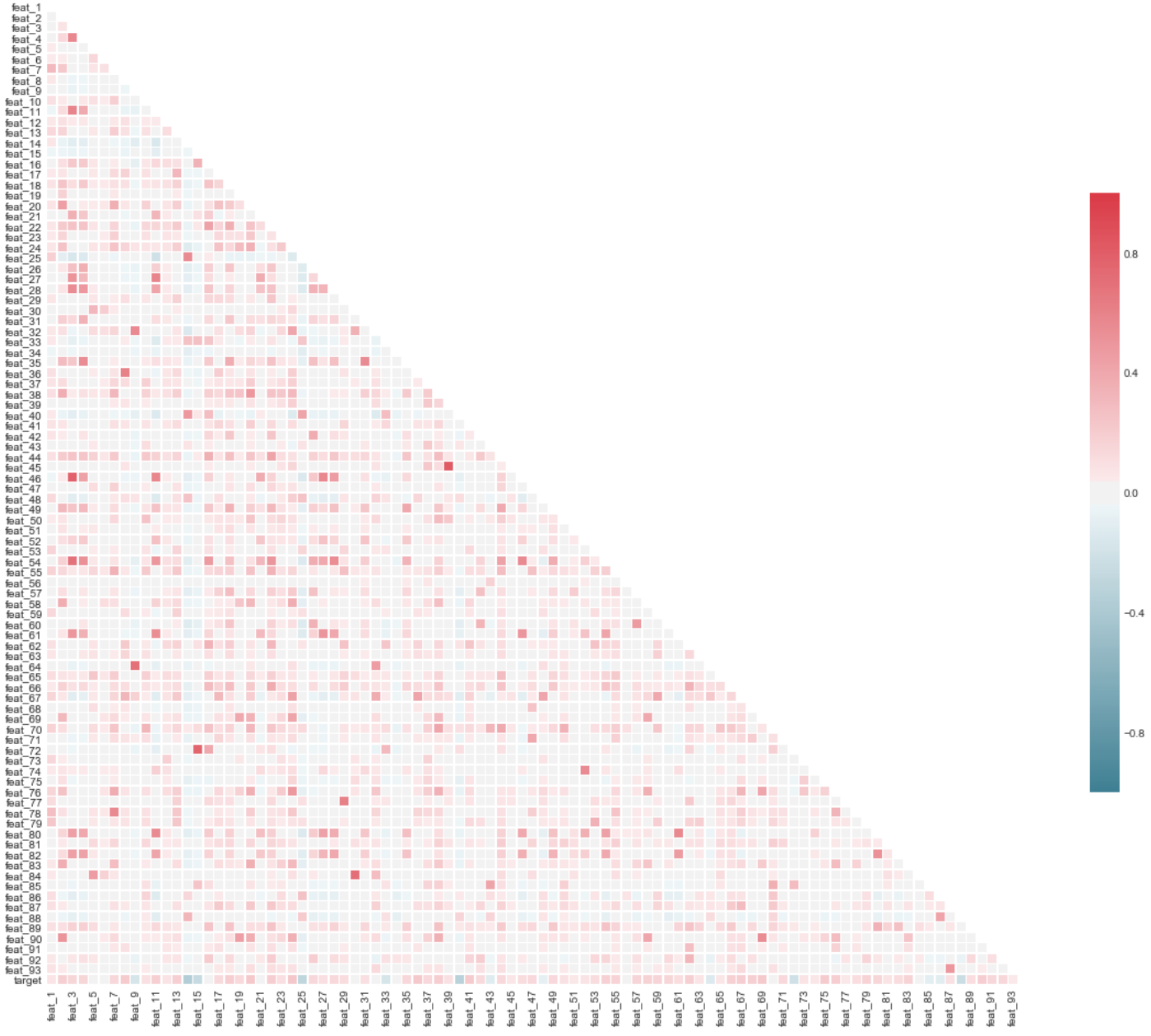
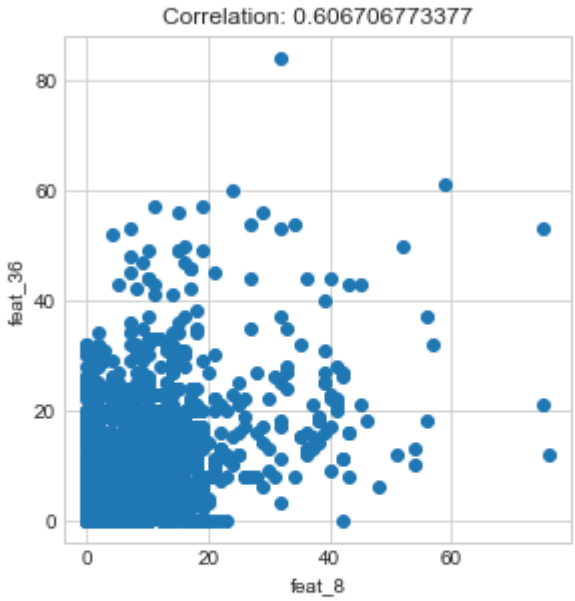
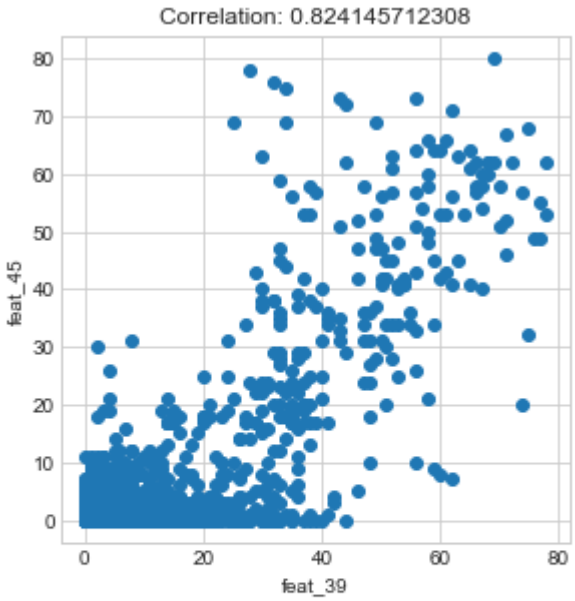
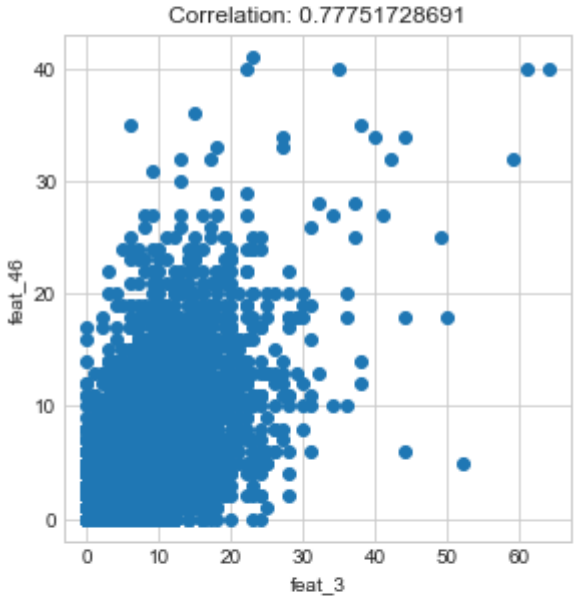


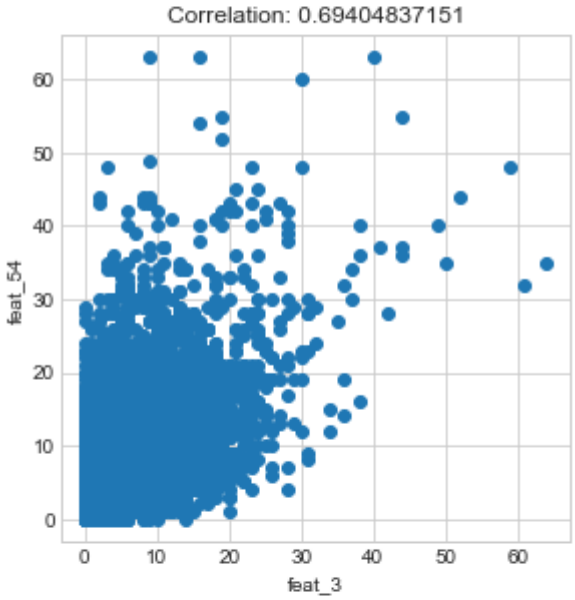
Fig 2. Highest correlation feature pairs

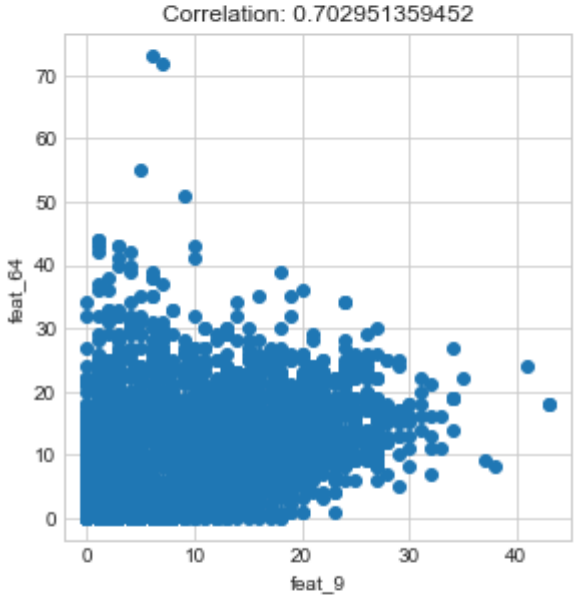
- [(8, 36), (39, 45), (3, 46), (3, 54), (9, 64), (15, 72), (29, 77), (30, 84)]

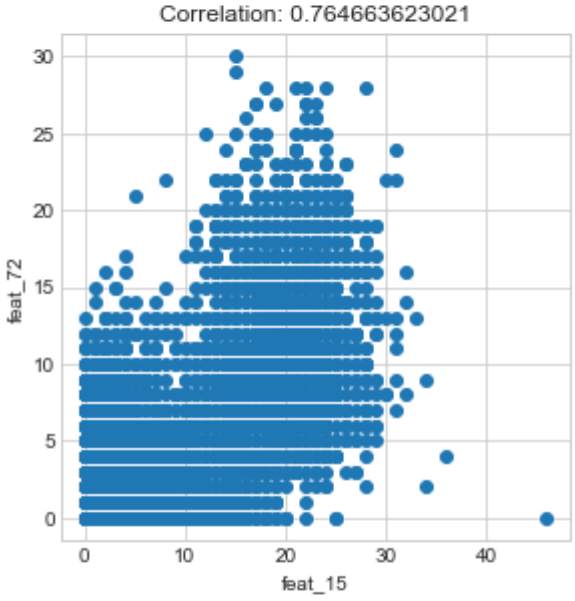

Correlation: 0.606706773377
feat_36
feat_8


Correlation: 0.824145712308
feat_45
feat_39

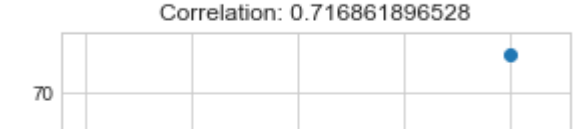

Correlation: 0.77751728691
feat_46
feat_3

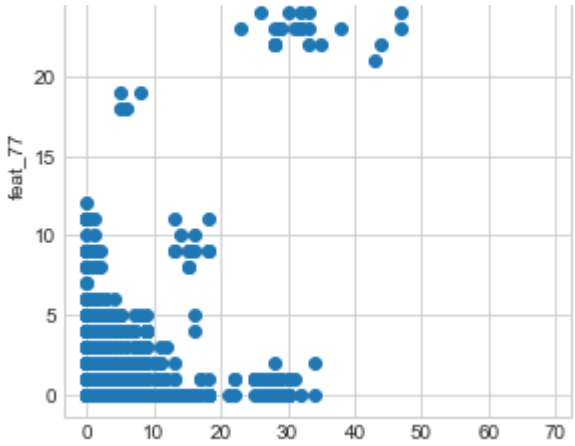

Correlation: 0.69404837151
feat_54
feat_3

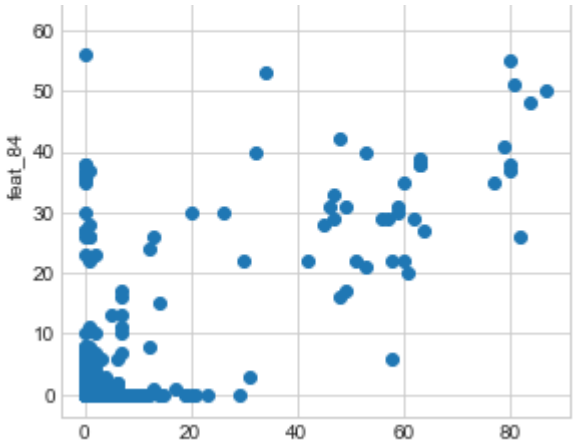

Correlation: 0.702951359452
feat_64
feat_9


Correlation: 0.764663623021
feat_72
feat_15


Correlation: 0.612847320817


Correlation: 0.716861896528





Are the correlations statistically significant?

Formally, the correlation coefficient, r, tells us about the strength and direction of the linear relationship between x and y. However, the reliability of the linear model also depends on how many observed data points are in the sample. We need to look at both the value of the correlation coefficient r and the sample size n, together. The hypothesis test lets us decide whether the value of the population correlation coefficient ρ is "close to zero" or "significantly different from zero". We decide this based on the sample correlation coefficient r and the sample size n.

Null Hypothesis: H0: $\rho = 0$

Alternate Hypothesis: Ha: $\rho \neq 0$

OR

Null Hypothesis H0: The population correlation coefficient IS NOT significantly different from zero. There IS NOT a significant linear relationship(correlation) between x and y in the population.

Alternate Hypothesis Ha: The population correlation coefficient IS significantly DIFFERENT FROM zero. There IS A SIGNIFICANT LINEAR RELATIONSHIP (correlation) between x and y in the population.

at significance level ($\alpha = 0.01$)

We will accept or reject the Null hypothesis based on the p values. We will apply this to all the feature pairs and find the highest correlated features. And we can compare that with our previous results too.

Result: The p values were equal to 0 and we rejected the null hypothesis that the population correlation is zero. The pairs of features [(8, 36), (39, 45), (3, 46), (3, 54), (9, 64), (15, 72), (29, 77), (30, 84)] are statistically linearly correlated at significance level of 1%

Important features to predict the target product category

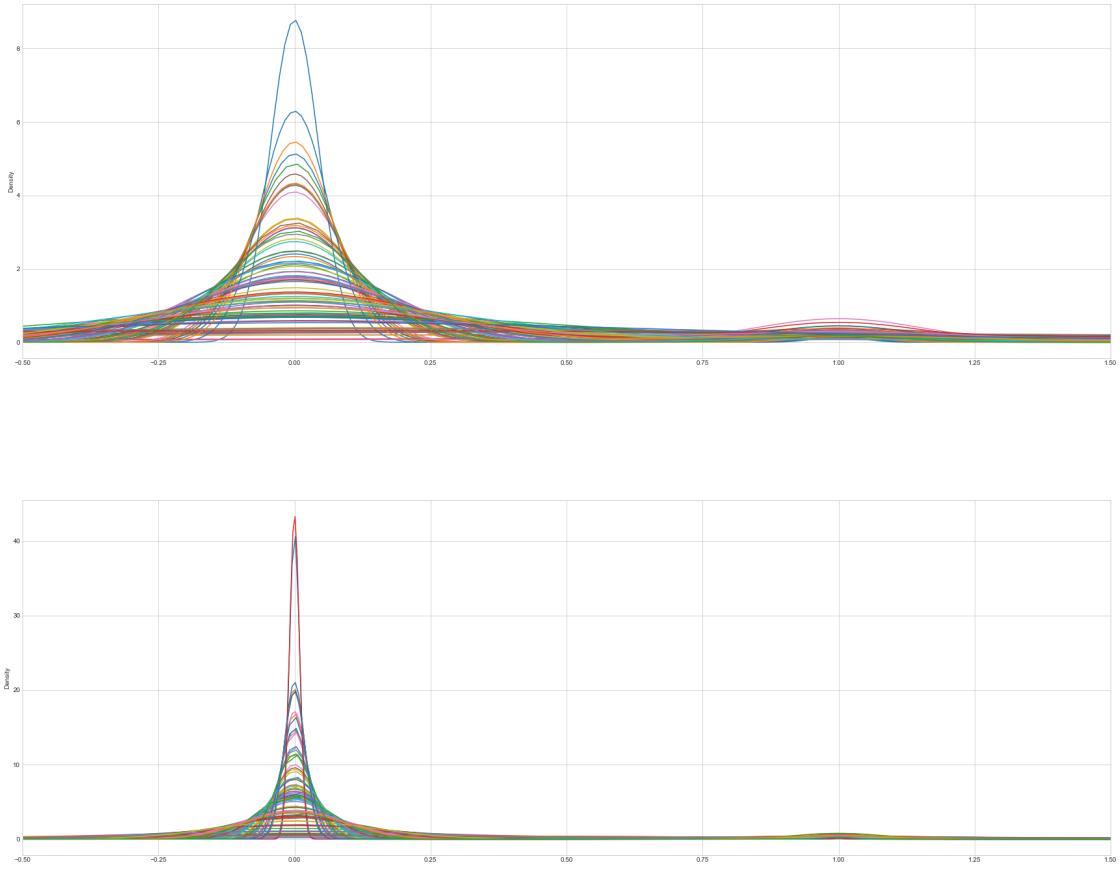
For a given feature if the difference between the classes is significant then that feature is important and should be used while finding out the best model.

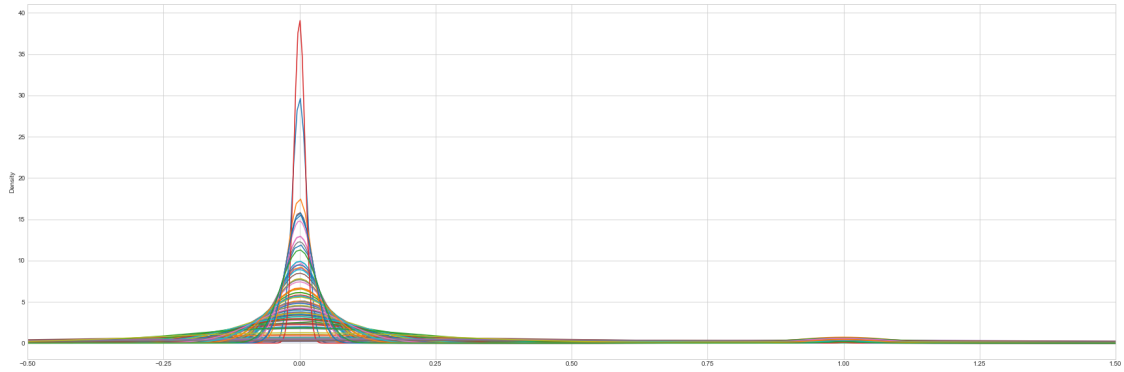
To find out the intra class difference for each feature we can utilize ANOVA. ANOVA can determine whether the means of three or more groups are different. It uses F-tests to statistically test the equality of means. However before we jump in doing ANOVA, there are certain assumptions that should satisfy:

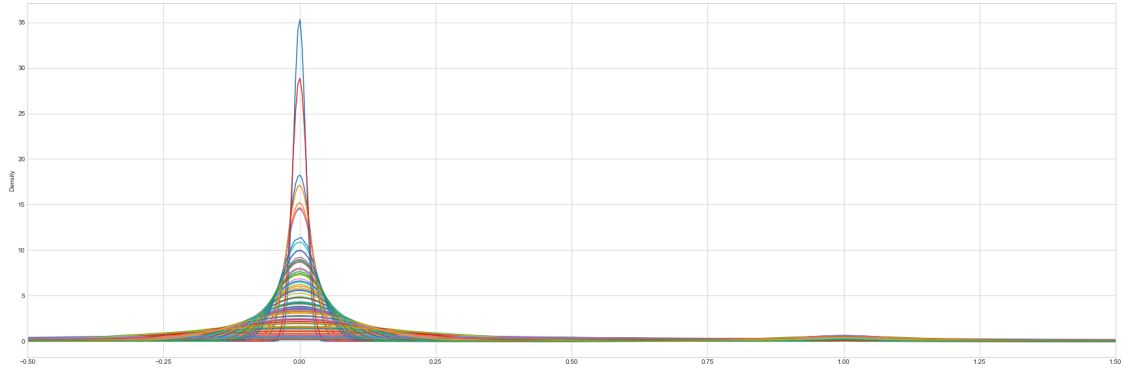
- The different populations should have same variances. This is called assumption of homogeneity of variance
- The different populations should be normally distributed
- Samples are independent

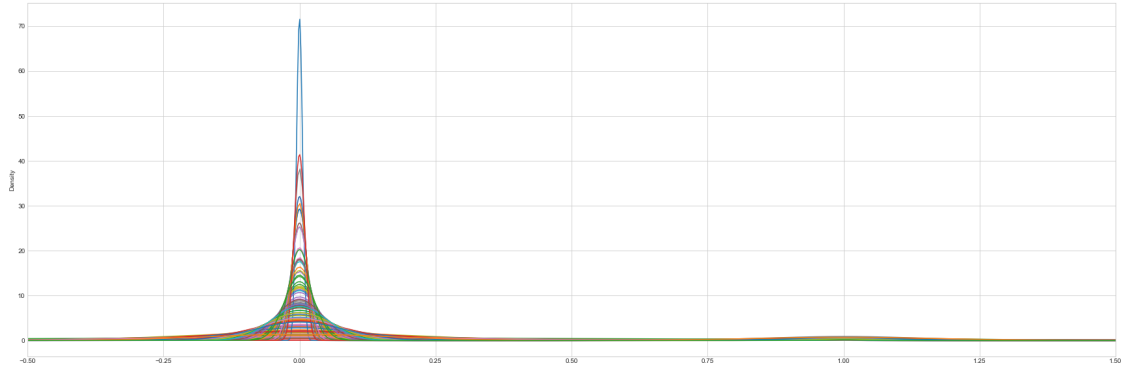
The 3rd assumption is apparently satisfied because there is no evidence in against of that. The first 2 assumptions can be relaxed a bit. Here is some visualizion of the data. The data spread is quite normal (with some skew) and similar spread/variance.

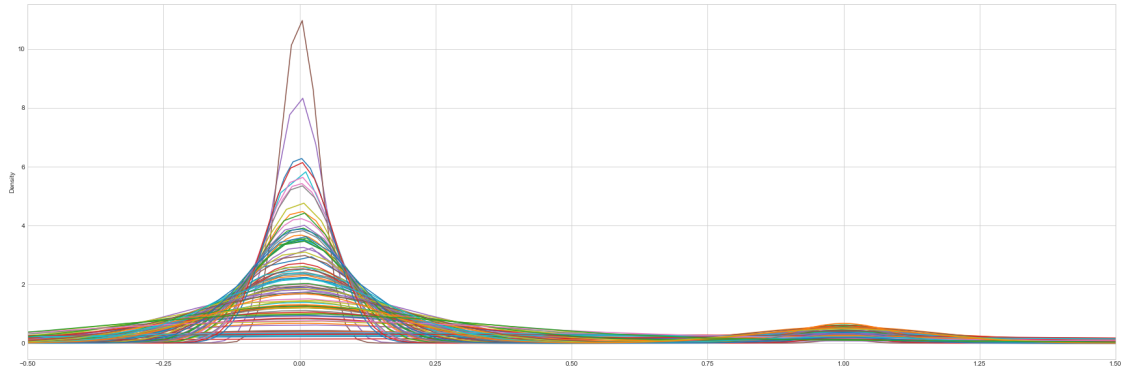
Fig 1. Features data spread for each class (1 to 9)

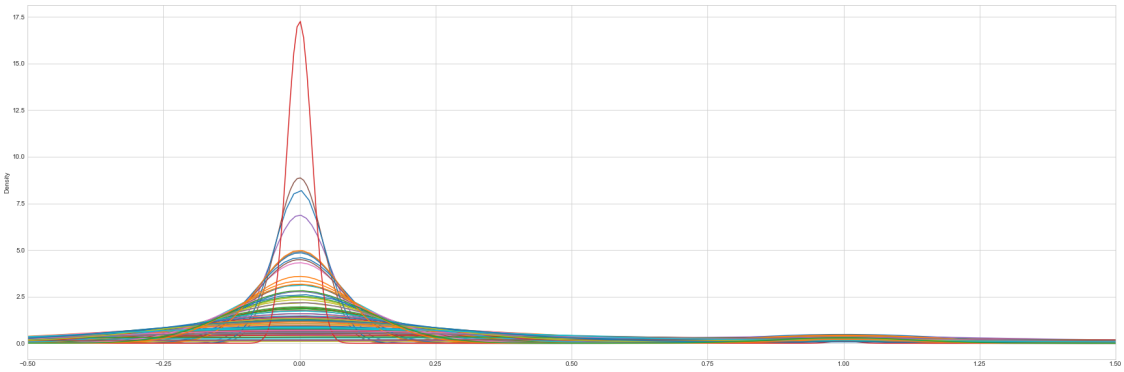


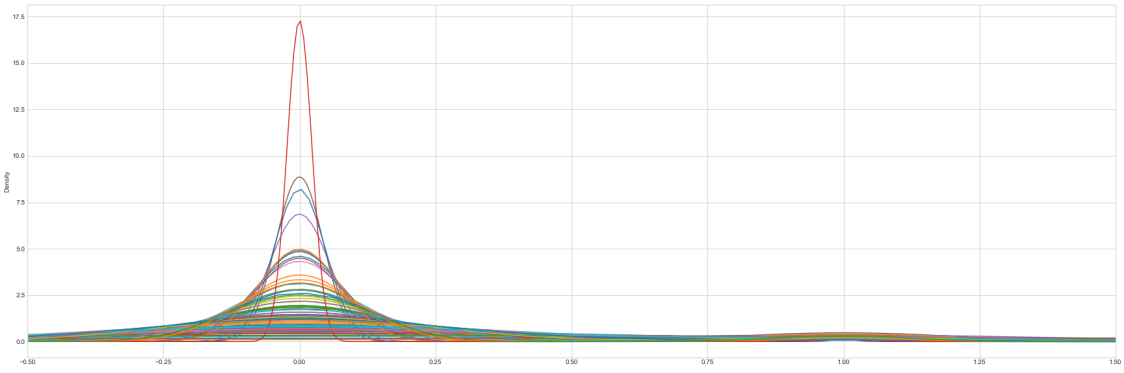

Density
40
35
30
25
20
15
10
5
0
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50

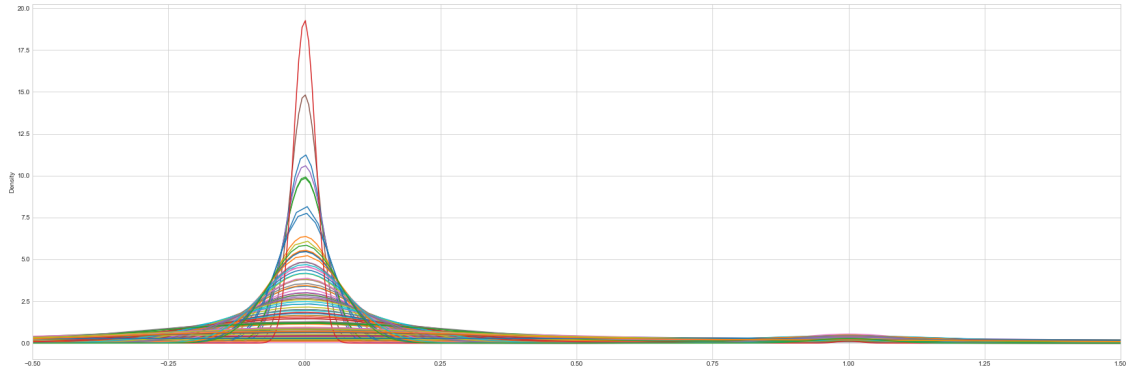

Density
35
30
25
20
15
10
5
0
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50


Density
70
60
50
40
30
20
10
0
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50


Density
10
8
6
4
2
0
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50


Density
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50


Density
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50



ANOVA and results

After applying the ANOVA for each feature following features came up as the important features: feat_34, feat_11, feat_14, feat_60, feat_25

Less imortant features: feat_65, feat_6, feat_51, feat_63, feat_12

Fig. Kernel density plots for top features

- features clearly differentiate between few classes
- for example, class 5 data is striking different for feature 11

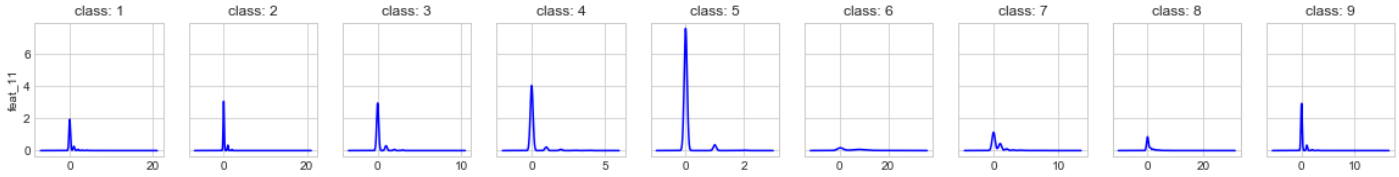
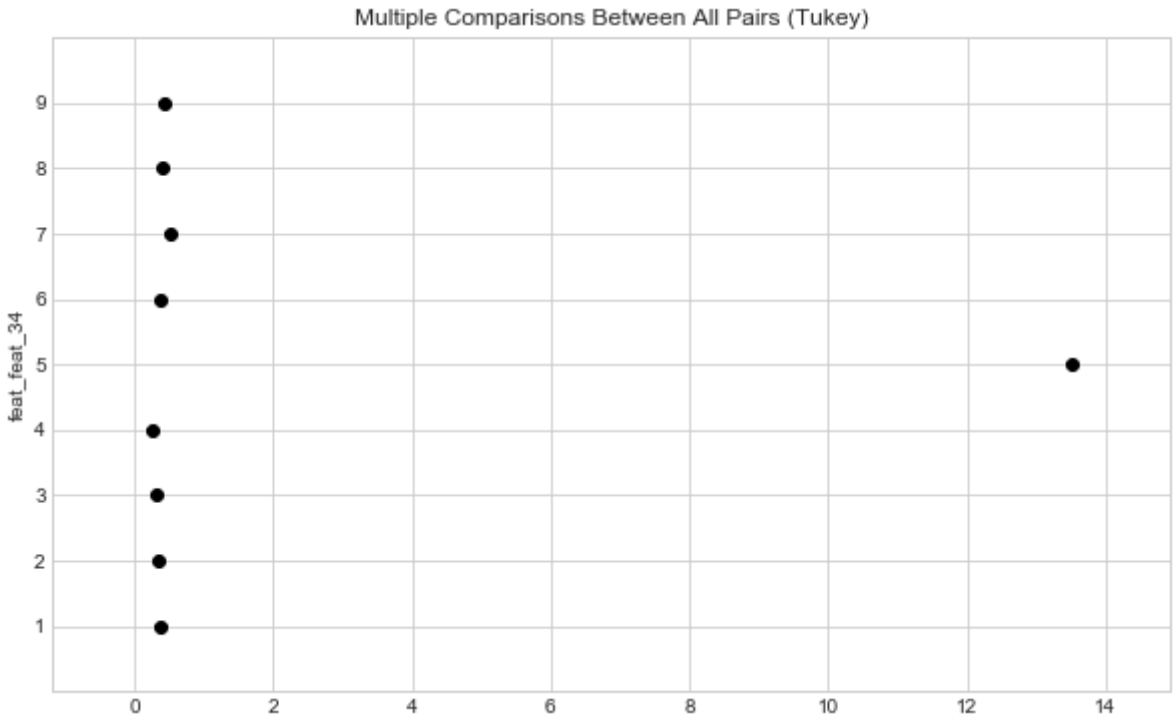


Fig. Tukey plots to visualize further



Features dimension reduction

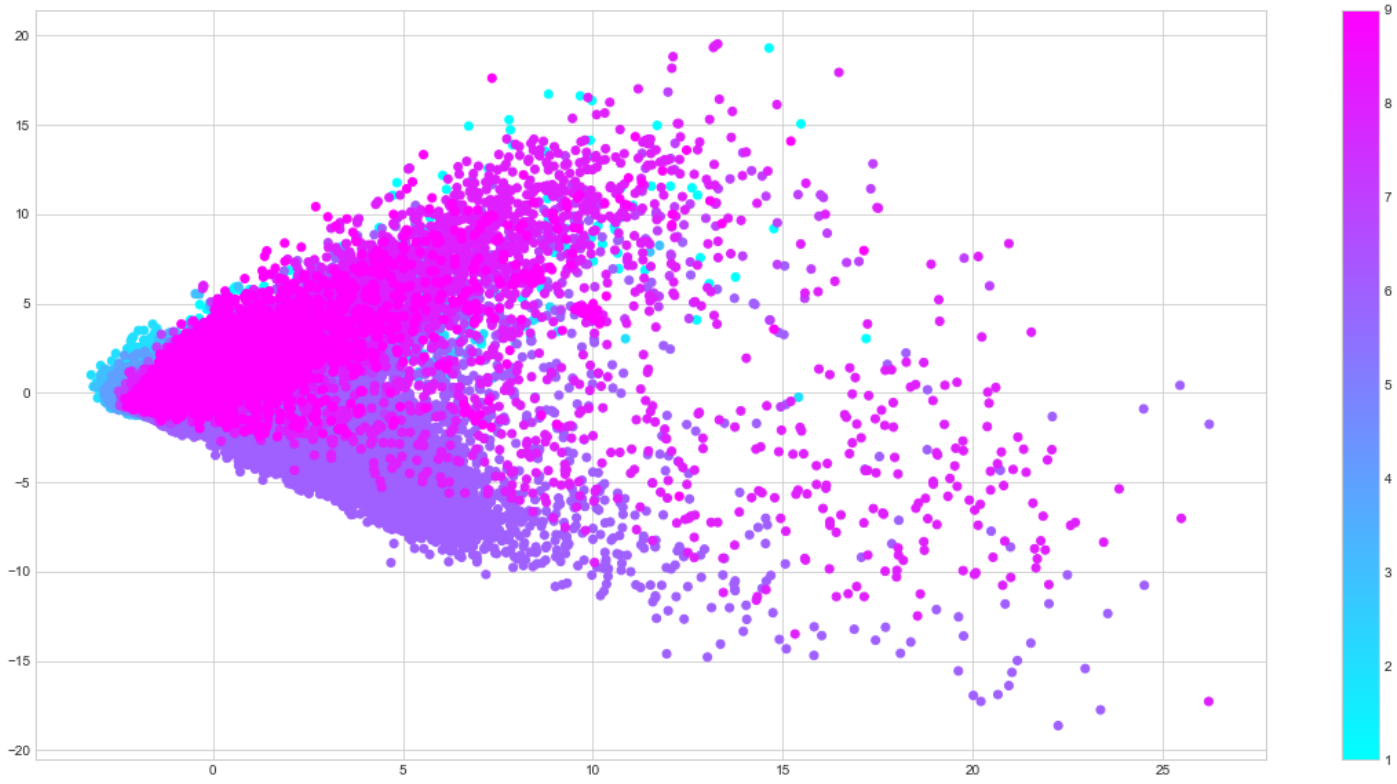
We have good amount of features. Although we could use all the features for our modeling. But if we can reduce the number of features by removing the redundant ones, then we could try more compute intensive models. PCA and LDA are some good techniques that we explore here. We will use subset of reduced dimension features while modeling.

PCA - Principal Component Analysis

The main idea of principal component analysis (PCA) is to reduce the dimensionality of a data set consisting of many variables correlated with each other, either heavily or lightly, while retaining the variation present in the dataset, up to the maximum extent. The same is done by transforming the variables to a new set of variables, which are known as the principal components (or simply, the PCs) and are orthogonal, ordered such that the retention of variation present in the original variables decreases as we move down in the order. So, in this way, the 1st principal component retains maximum variation that was present in the original components.

Fig. First two PCA features

- First 35 pca components explain the 70% of the variation within the data.
- This is not a great result but we will see if that helps in our modeling.

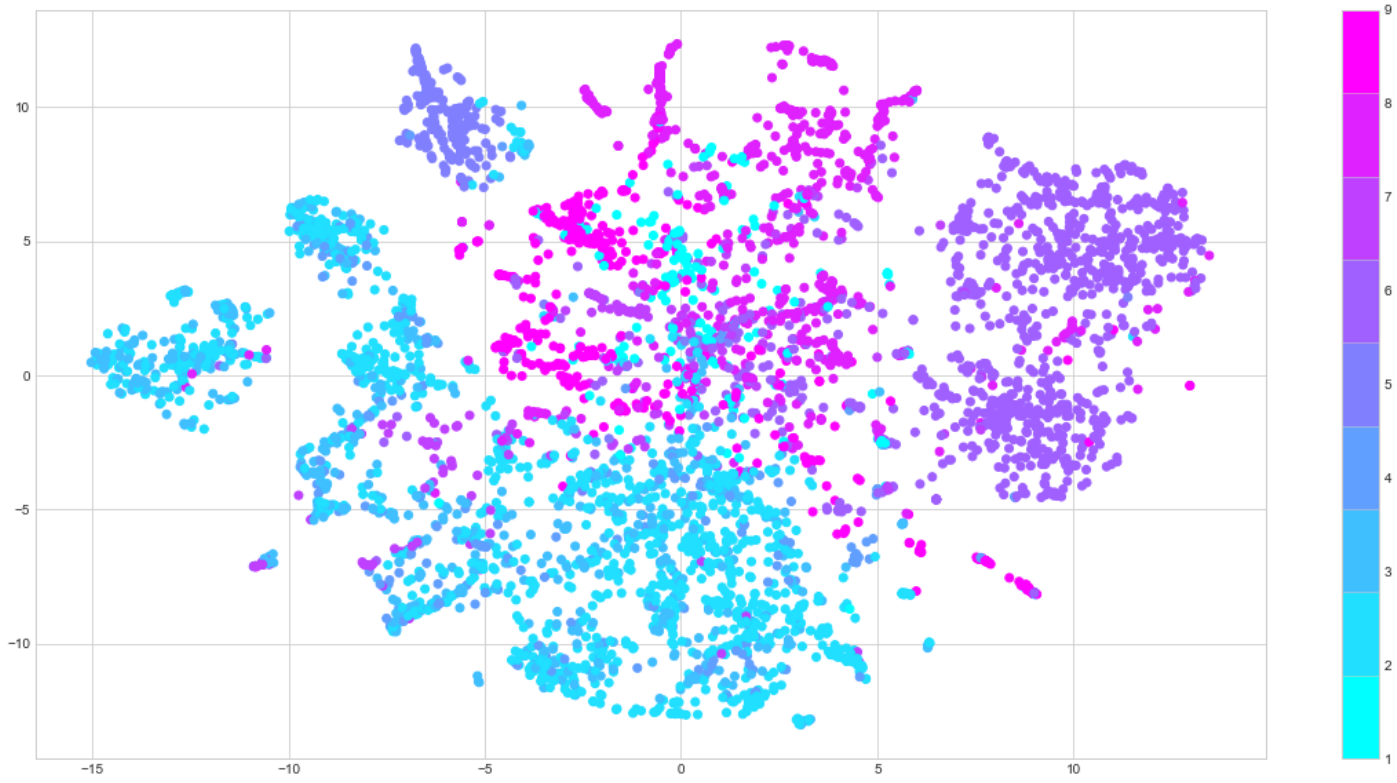


T-SNE (T-Distributed Stochastic Neighbouring Entities) visualiztion

T-SNE is another technique for dimensionality reduction and is particularly well suited for the visualization of high-dimensional datasets. Contrary to PCA it is not a mathematical technique but a probablistic one. It finds patterns in the data by identifying observed clusters based on similarity of data points with multiple features. It is mainly a data exploration and visualization technique. But t-SNE can be used in the process of classification and clustering by using its output as the input feature for other classification algorithms. In our case we are mainly interested in visualizing the data.

Fig. First two t-SNE features

- based on 8000 data points only because its compute intensive
- it gives good sense that the features are able to distinguish between different classes

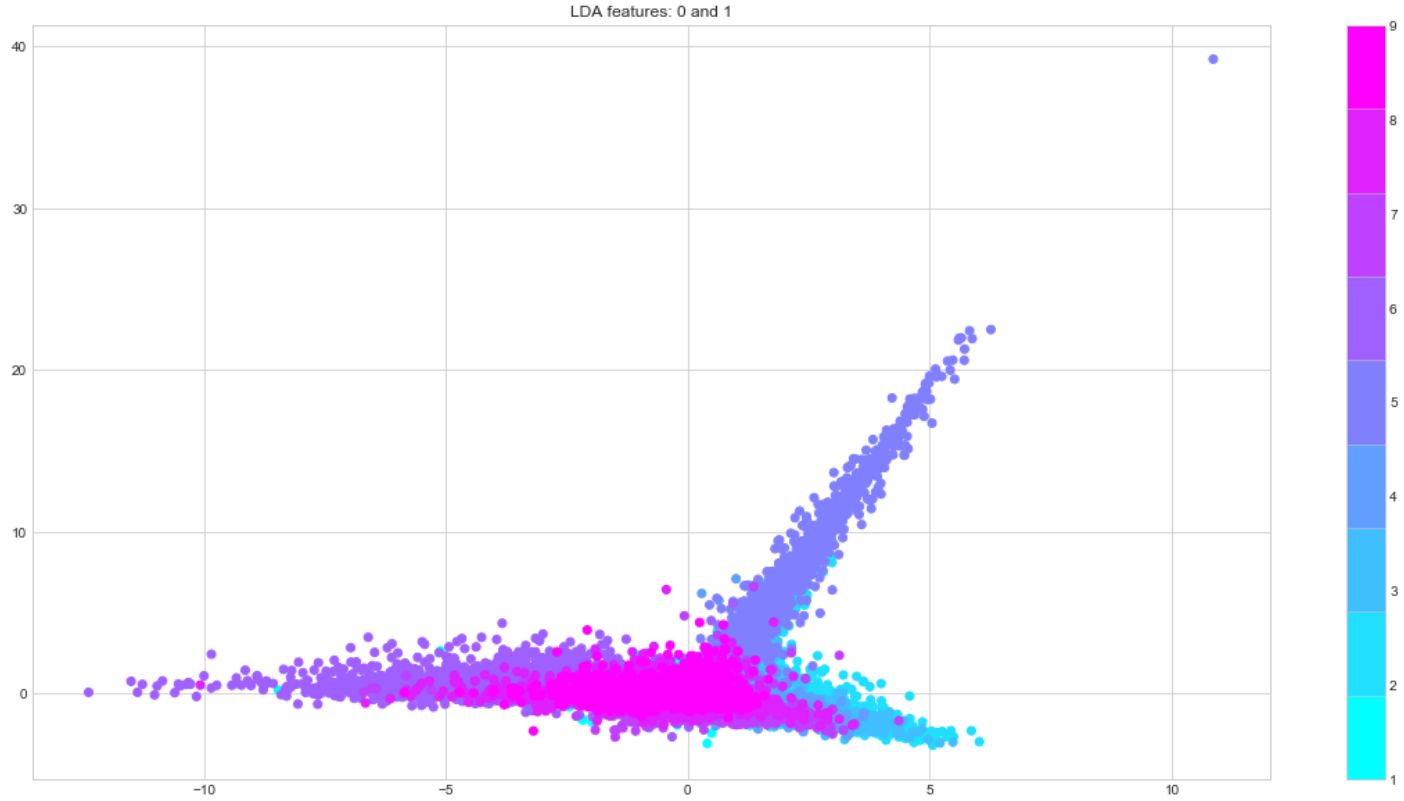


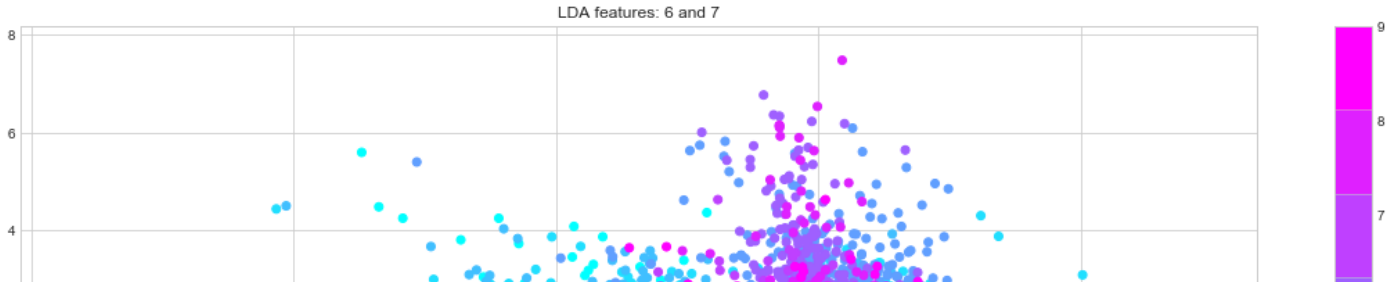
LDA - Linear Discriminant Analysis

LDA can be used as both a dimensionality reduction technique and a classifier. LDA basically separates example of classes linearly moving them to a different feature space, and therefore if the dataset is linearly separable, applying LDA as a classifier will get great results. However in this case, the dataset is not linearly separable, so the LDA will try to organize your dataset in another space as the maximum linearly separability as possible, but there will still be overlapping between classes because of non-linearly characteristic of data. We can apply different classifier on top of LDA features.

Fig. LDA features visualization

- LDA features separate the classes fine and could be used as features for classifier
- Features 6 and 7 seem to have lots of overlap





Modeling

Theory and Approach followed

Our goal is to develop a model that describes the relationship between the features in the data and the target class (product category). So far we have got good understanding of the data and we can apply the machine learning algorithms and find out the best model. Technically speaking, this is Supervised Classification ML problem. Here we have the training data with the actual labels. We need to model that and use the model to predict labels for new data.

We will start with a quick and dirty baseline model. And improve upon our model in interative fashion. Following are some ideas to keep in mind.

Overfitting and Underfitting

When the model fits the training data too perfectly but not the test data, it is called overfitting. Such a model will not be useful for predictions. While at the other extreme end our model might fit the training data very poorly. Consequently, it can't fit test data any better and will give inaccurate predictions. Generally its a trade off between overfitting and underfitting. This is also called **Bias-Variance trade-off**. When the model fits too well to the training data (a subset of population) it has low Bias, that is, it mimics the data very well. But it might fail miserably in mimicking the test data (another subset of population). Or it has variance between different subsets of data. As we try to make this variance low, it will increase the Bias.

In our case, the number of data points is very large (67 thousand) as compared to the number of features. This will help in avoiding overfitting. We are going to always compare train and test accuracy to see how well the model fits. We should also design train-test split of the data wisely to regulate the bias-variance trade off.

Model Evaluation Criteria

Evaluating our model closely relates with how the model is going to be used by the client. In case of classification problems one type of error (say false negative) might be more costly then another type of error (false positive). In such a case our model should be more stringent for false negatives.

In our case we need to classify each product (described by 93 features) into correct class. As such both type of errors have equal weightage. One important fact is that our traning data has class imbalance. There is more data for some classes then others. This is important consideration while chosing the model evaluation criterion. We should try and handle the class imbalance problem and see if improve our models. Criteria that we are going to use.

- **Accuracy**: Although important we can't rely on just that. Because we have class imabalnce, our accuracy might be high but predictions for minority class might be wrong.
- **Confusion Matirx**: This would tell us how many precdictions we got wrong for each class. And how well the minority classes are doing along with majority classes.
- **Classification Matrix**: To measure precision, recall and F1 score (harmonic mean of precision and recall) for each class. It tells us average F1 score for all classes too. This can serve as a single value to compare the models.
- **Log loss**: This is the evaluation criteria used by the client (Otto Group). So this is our main criterion to compare and improve the models. Each product has been labeled with one true category. For each product, we find a set of predicted probabilities (one for every category). The formula is then,

\begin{equation*}

$$logloss=-\dfrac{1}{N} \sum_{j=1}^M \sum_{i=1}^N y_{ij}log(p_{ij}),$$

\end{equation*} where N is the number of products in the test set, M is the number of class labels, log is the natural logarithm, yij is 1 if observation i is in class j and 0 otherwise, and pij is the predicted probability that observation i belongs to class j.

Data and code organization

- **Data files generated and saved instead of data pipelines in the runtime**

  Although data pipelines are generally used transform train and test data sets. In this project the data is huge and we don't want to run all the steps of datapipeline while development. Moreover there is not much of data transformation. Mainly we do the Standard Scaling and save the resulting data as a file int he /data folder. This file is used for training of most the models. Similiarly the cleaned data files are also saved into the /data folder.

- **Python functions written to apply the DRY principle**

  Since lot of things are repetitive like fitting the data, evaluating the results etc. we have written Python functions are used multiple times.

- **Fitted Models are also saved**

  Again because the data is huge we don't want to run the (timeconsuming) model fitting whenever we restart the Jupyter notebook. So we do save the fitted models in the /model-repo directory.

First (Baseline) Models

Baseline Results

Each machine learning problem is unique and before embarking on trying out different algorithms and strategies, we should first find out the baseline result. A baseline result is the simplest possible prediction. For e.g. if we are predicting the prices of houses, then the baseline would be the average of the train set prices. The baseline will serve as a comparison point for all our future results. In case of multi-class classification proble that we have we can calculated the baseline results as follows:

Accuracy: 0.26050

Predicting all the target classes as "2" because that is the most occuring class in our train data set.

Log loss: 25.54128

We predict the probability for class "2" as 1 and all other class probabilities as 0s for all the data in our train set.

Given the base result we would expect our Machine Learning models to give better results than this. If not, then we may need to collect more or different data from which to model and perhaps more powerful machine learning algorithms. If that doesn't help we may have a problem that is resistant to prediction and may need to be re-framed.

Held out test set

The given data set is divided into two parts 99% (55690 rows) for training and validation and 1%(6188 rows) as final test set. The reason for only 1% here is that we have a huge data set.

We shall fit various algorithms on the train set. This includes dividing the train set into validation set when we need to train hyper parameters.

Logistic Regression

We start with the basic machine learning model for linear classification i.e. Logistic Regression. All the linear models divide the data by linear surfaces to classify. We start with the default Logistic Regression model provided by the sklearn library and go on to try different parameters and hyper parameters.

Parameters: **Algorithms One vs All and Multinomial.** OVA/OVR works by fitting LR models for each of the class. At the time of prediction, the target class is chosen based on which model that gives the highest probability. I also tried the OVR approach manually. It produced similar results. Multinomial algo uses cross-entropy loss function.

Regularization L1 and L2 Regularization is a very important technique in machine learning to prevent overfitting. Mathematically speaking, it adds a regularization term in order to prevent the coefficients to fit so perfectly to overfit. The difference between the L1(Lasso) and L2(Ridge) is just that L2(Ridge) is the sum of the square of the weights, while L1(Lasso) is just the sum of the absolute weights in MSE or another loss function.

Also different strength of regularizations are tried with C parameter.

Observations: We see that parameter combination (saga, l1, multinomial, C:10) and (lbfgs, l2, multinomial, C:100) work better than others. The log loss (0.63) is less and the F1 score (0.75) and accuracy (0.76) is more. And we see that all the models log loss and accuracies are better than our baseline results.

Table 1. Logistic Regression with different params

Parameters	Hyperparams	Test Results				
		Log Loss	Precision	Recall	F1	Accuracy
liblinear, l1, ovr	C:10	0.66987	0.76	0.76	0.73	0.75614
saga, l1, ovr	C:10	0.67156	0.76	0.76	0.73	0.75549
saga, l1, multinomial	C:10	0.63743	0.76	0.76	0.75	0.76471
lbfgs, l2, ovr	C:10	0.67034	0.76	0.76	0.73	0.75501
lbfgs, l2, multinomial	C:100	0.63614	0.76	0.77	0.75	0.76568

Dealing with classes imbalances

Machine Learning Algorithms are usually designed to improve accuracy by reducing the error. Thus, they do not take into account the class distribution / proportion or balance of classes. If the data is not balanced for all the classes, the minority class will have less effect on the model we develop. This may be a problem specially when the minority class is important and we don't want to miss predict for that class. Simple example is of fraud detection in case of credit card transactions. 99% or more times we have genuine transactions and our model can do all predictions as non fraud and be 99% accurate. But that would be useless.

In our case, all classes are equally important but we have imbalanced data. So I balance the data to try our new models. At high level there are 2 ways to deal with class imablance. (1) Balance the data and then apply ML algorithms. (2) Take care of class balancing at algorithm level.

(1) Balancing the data

Oversampling: The basic logic is to apply some technique to generate more samples for the minority classes so that the size of the data from all the classes is equal.

Random Over Sampling - It increases the number of instances in the minority class by randomly replicating them.

SMOTE - Synthetic Minority Over sampling Technique (SMOTE) algorithm applies KNN approach where it selects K nearest neighbors, joins them and creates the synthetic samples in the space. The algorithm takes the feature vectors and its nearest neighbors, computes the distance between these vectors. The difference is multiplied by random number between (0, 1) and it is added back to feature.

ADASYN - ADAptive SYNthetic (ADASYN) is based on the idea of adaptively generating minority data samples according to their distributions using K nearest neighbor. The algorithm adaptively updates the distribution and there are no assumptions made for the underlying distribution of the data.

Undersampling: The basic logic is to reduce the data from the majority classes so as to make all the data from all the classes equal.

Prototype Generation methods:

Prototype generation technique will reduce the number of samples in the targeted classes but the remaining samples are generated — and not selected — from the original set.

ClusterCentroids - Each class will be synthesized with the centroids of the K-means method instead of the original samples.

Prototype Selection methods:

Prototype selection algorithms will select samples from the original set.

Random Under Sampling - It is a fast and easy way to balance the data by randomly selecting a subset of data for the targeted classes.

NearMiss - It addds heuristic rules to select samples. These rules are based on nearest neighbors algorithm. NearMiss-1 selects the positive (majority class) samples for which the average distance to the N closest samples of the negative class is the smallest. NearMiss-2 selects the positive samples for which the average distance to the N farthest samples of the negative class is the smallest. NearMiss-3 is 2 step. First for each negative class sample their M nearest positive class neighbors will be kept. Then the positive samples will be kept for which the average distance to the N nearest neighbors is largest.

ENN - EditedNearestNeighbors applies a nearest-neighbors algorithm and “edit” the dataset by removing samples which do not agree “enough” with their neighboorhood. For each sample in the class to be under-sampled, the nearest-neighbours are computed and if the selection criterion is not fulfilled, the sample is removed. Two selection criteria are currently available: (i) the majority (i.e., kind_sel='mode') or (ii) all (i.e., kind_sel='all') the nearest-neighbors have to belong to the same class than the sample inspected to keep it in the dataset.

IntanceHardnessThreshold - It is a specific algorithm in which a classifier is trained on the data and the samples with lower probabilities are removed. It needs estimator parameter. We have used Logistic Regression as the estimator.

Combination of oversampling and undersampling

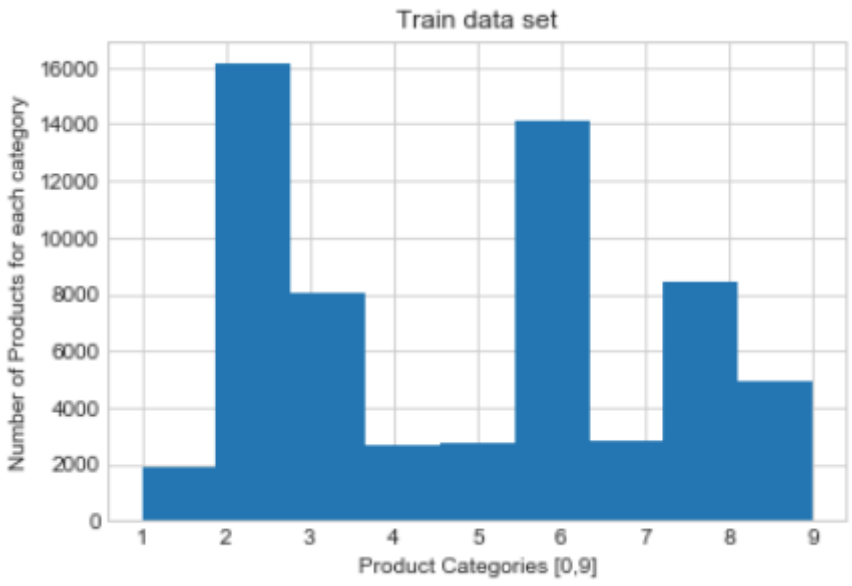
SMOTEENN - SMOTE method can generate noisy samples by interpolating new points between marginal outliers and inliers. This issue can be solved by cleaning the space resulting from over-sampling. It the ENN is used for undersampling/cleaning it gives SMOTEENN.

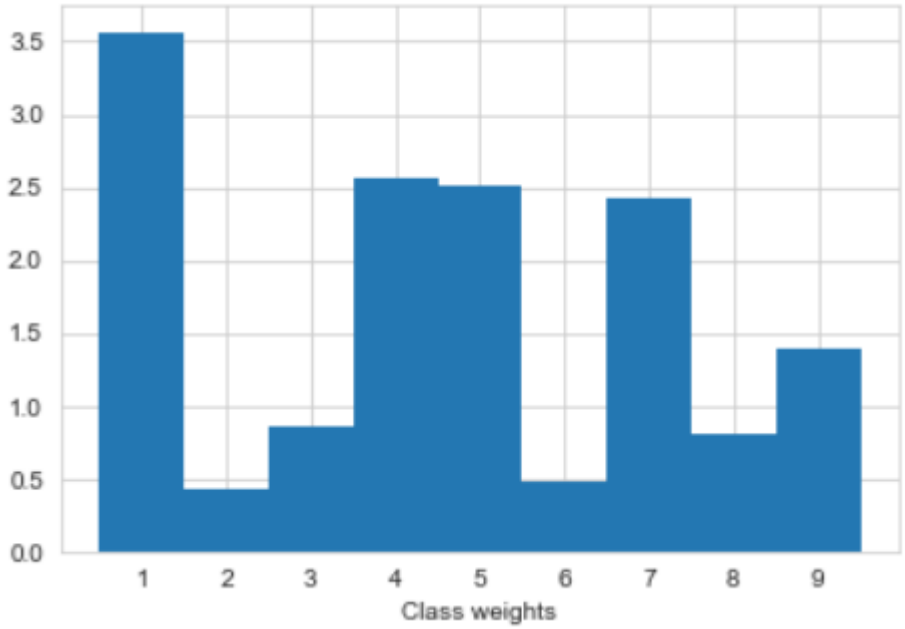
(2) Taking care of class balancing at algo level

Weights applied while running the Algorithm

Weights were applied inversely proportional to the number of rows for each class. Parmeter used is class_weight={1: 3.56, 2: 0.42, 3: 0.85, 4: 2.55, 5: 2.51, 6: 0.48, 7: 2.42, 8: 0.81, 9: 1.38} for LogisticRegression.

Fig. Class distribution and chosen class weights




3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
1
2
3
4
5
6
7
8
9
Class weights

Observations: After applying different methods of balancing the classes, we see that there is not much improvement provided by any of the balancing methods. Infact the performance is worse than without balancing the classes. Hence we can ignore the class balancing part and move on to other ML algorithms. Here are the results for the class balancing methods.

Table 2. Results after balancing the classes

Class Balacning Method	Test Results				
	Log Loss	Precision	Recall	F1	Accuracy
Normalized, Naïve Random Over Sampling	0.79605	0.77	0.71	0.73	0.71057
Normalized,ADASYN Over Sampling	0.67975	0.76	0.76	0.73	0.75663
Normalized,ADASYN Over Sampling	0.8348	0.75	0.69	0.7	0.68714
SMOTE random over sampling	0.78492	0.77	0.71	0.72	0.70814
Prototype generation under sampling	0.99595	0.76	0.59	0.62	0.58759
Random under sampling	0.81376	0.77	0.71	0.73	0.71202
Near miss under sampling - version1	2.35544	0.79	0.63	0.7	0.54945
Near miss under sampling - version2	1.03002	0.72	0.57	0.59	0.57143
Near miss under sampling - version3	0.89183	0.72	0.7	0.71	0.70023
EditedNearestNeighbours	0.82391	0.76	0.75	0.73	0.7521
InstanceHardnessThreshold	1.91493	0.72	0.7	0.71	0.52666
SMOTEENN under and over sampling	1.07592	0.78	0.64	0.65	0.63849
Weighted classes (weights inversely proportional to the number of rows for each class)	0.73284	0.76	0.75	0.75	0.75323
Logistic Regression with one vs all approach (9 times for each class)	0.67151	0.76	0.76	0.73	0.75533
Logistic Regression with one vs all approach (9 times for each class) - Balanced	0.94957	0.75	0.73	0.74	

Other simple Models

Logistic Regression was great improvement from the Baseline results, meaning our data has signal and capacity to classify the target class. We also observed that balancing the data didn't help much. Now we try different models to see if we can get any better results. Mainly we are focusing on the log loss metric.

MultinomialNB

Naive Bayes is a family of algorithms based on applying Bayes theorem with a strong(naive) assumption, that every feature is independent of the others, in order to predict the category of a given sample. They are probabilistic classifiers, therefore will calculate the probability of each category using Bayes theorem, and the category with the highest probability will be output.

MultinomialNB assumens that the feature data come from a multinomial distribution. The multinomial distribution describes the probability of observing counts among a number of categories, and thus multinomial naive Bayes is most appropriate for features that represent counts. This looks similar to our case where the features represent some counts.

The log loss result of 3.45 is very poor and it looks like that our data does not have clean separation and we need other complex models.

Support Vector Machines

SVMs deal with finding the hyperplane in the n-dimensional feature data space that separates the classes very well. It applies the concept of maximum margin between the hyperplane and nearest data points (support vectors). This classifier words well in cases of large number of features. The training time is quite high in our case because there is lot of data. But once trained, it only saves few points as support vectors and hence consumes less memory at the time of prediction.

The log loss result is 0.56, which is best so far. Let us explore more models, but keep this in mind to be used for ensemble etc.

K Nearest Neighbors

KNN is non-parameteric classifier. It keeps all data points as the model and predicts target for new data points by checking out the nearest neighbors. It is also called lazy because it calculates everything at the moment of prediction. So training is fast but prediction is slow for huge data set. To calculate the distances from the nearest neighbors it uses Euclidean or Hamming disctances. In our case all features are numeric and default of Euclidean is fine.

The log loss result is 1.43, which is only better than MultinomialNB. The best value of number of neighbors found by GridSearchCV was 5.

Table 3. Results for Other models

Model	Parameters	Log Loss Result
MultinomialNB	alpha=1.0	3.45009
SVC	C=1.0, gamma='auto', kernel='rbf'	0.56515

Ensemble Models

Ensemble learning uses hundreds to thousands of models of the same algorithm that work together to find the correct classification. The ones that are very common use the tree based DecisionTree classifier as the base model.

Bagging Models - Random Forest

For Bagging, different samples are taken from the data with replacement and model hypothesis are generated for each sample. Once each model has developed a hypothesis, the models use voting for classification or averaging for regression. All hypothesis have equal weight.

Here we use RandomForest that uses DecisionTree as the base classifier. Random Forests are also very hard to beat in terms of performance and I saw the similar performance.

The log loss result is 0.9, which is not bad as compared to others.

Boosting Models - Gradient Boosting

Boosting is an ensemble technique in which the predictors are not made independently, but sequentially. This technique employs the logic in which the subsequent predictors learn from the mistakes of the previous predictors. Because new predictors are learning from mistakes committed by previous predictors, it takes less time/iterations to reach close to actual predictions. For Gradient Boosting, typically Decision Trees are used as predictors.

The log loss result is 0.47, best result so far.

Boosting Models - Extreme Gradient Boost

XGBoost also applies Gradient Boost algorithm but with an additional custom regularization term in the objective function. It is really fast when compared to other implementations of gradient boosting. Also it is one of the most popular ML classifier.

Stacking Models

We also tried other ensemble models which can use multiple types of classifiers as base. And we tried little bit of stacking. Stacking refers to different layers of applying models. The results from first layer of models are used as feature for the next layer. In particular we applied KNN, LogisticRegression and RandomForest at first layer and LogisticRegression at the second layer.

The log loss result of 1.34 wan't particularly good, but we have lot of scope to try different combinations. We leave that for future analysis.

Table 4. Results for Ensemble and Stacking models

Model	Parameters	Log Loss Result
RandomForestClassifier	max_depth=10, max_features=20,n_estimators=500	0.90412
GradientBoostingClassifier	max_depth=5, max_features=20, n_estimators=1000	0.49744
as (Logistic, GaussianNB, SVC) and meta classifier as LogisticRegression		1.34294
as (KNN, Logistic, RandomForest) and meta classifier as LogisticRegression		1.89438

Model Comparisons

To get the best signal out of the given data we explored various models and approaches. Since the data given was imbalanced and unequal distribution of instances from different product categories (classes), a lot went into balancing the classes both on data and algo level. Unfortunately it didn't give better models. Or other way to think of it is that the given data is works fine without balancing the classes. So we abandoned balancing and moved on without it.

There was not much scope for feature engineering because of the lack of domain knowledge. The data doesn't have meanings for features or the classes. Nevertheless, we tried adding some new features like maximum value, total of all values and count of all zero/non-zero values among given features. These features made the predictions worse.

Since the train data was huge and compute resources limited, we also tried taking subset of data to train some of the models faster. It helped mainly to quickly find the best hyperparamters, which were then used to train the whole set.

Dimensional reductionality was also utilized for some models. For e.g. PCA features were used. Also tried to eliminated redundant features found during data exploration phase. But turned out using all the features gave best results.

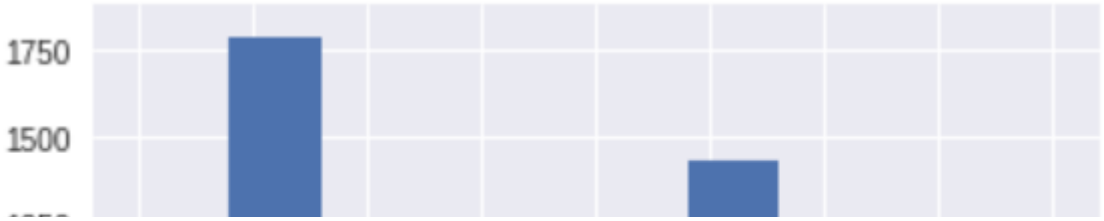
The **best model is the GradientBoost** followed by SVM, RandomForest and LogisticRegression. MultinomialNB and KNN did worst prediction on our held out test data.

Summary of GradientBoost model results:

- Log Loss: 0.49
- Accuracy: 82% of predictions on our test data was correct.
- Precision: 0.82
- Recall: 0.82
- F1: 0.82

Following chart shows that this model predictions are in line with the actual class distribution.

Fig. Class distribution for our test results



Model Recommendations

For making future predictions based on the analysis done so far, GradientBoost model can be used. The model is saved in the /models-repo directory and can be loaded and directly utilized to do the new predictions.

Steps to do predictions (refer the run_predictions.ipynb):

1. Import the joblib from sklearn.externals
2. Load the gradient_boost_model.sav file
3. Do the predictions using the model for new data points

Conclusions and Future analysis

The problem of Otto group product classification turned out to be a hard challenge. The data seems to be very unique in several senses. The signal from the data seems to be evenly spread. No single feature is contributing hugely as compared to others.

Other difficult thing is that the data has no meaning attached. It is just numbers, so it is hard to do any reasonable feature engineering.

To get insight into the data we did lots of visualizations. It was clear that our data had good amount of signal and it would be beneficial to try out various machine learning techniques to find that out. We took help of statistical analysis as well to find out how features were correlated and if they really helped distinguishing one product category from another. Applied various techniques like balancing the classes data, cross validation for finding our best hyper parameters, dimension reduction techniques to reduce the number of features and better visualization.

In future if we get more data, we can update our model by training it again. But it would really help to improve our models if we know the feature and product category names and meanings. The will aid in finding new features and better interpretation of the models.

If we get more compute resources, stacking and ensembling models with different combinations should be tried out. Last but not least we can try Neural network ML models. They tend to give good results when the data and number of features is huge as in this case.