Definition

Project Overview

Instacart is a United States based, same-day grocery delivery company that allows you to shop from your local grocery stores online, and have them delivered within 2 hours. The Instacart data science team is currently trying to use transactional data to model and predict the products users will reorder, try for the first time, or add to their cart during a session. For this reason, they have anonymized and open-sourced over 3 million orders, and hosted a Kaggle competition to challenge Kagglers to predict the products that customers will reorder (link to Kaggle competition here).

The dataset provides information on orders including the products purchased, hour of day and day of week for the order, days since last order, and order in which the items were added to the basket (as well as several other features). It also provides information about the products, including aisle and department. Finally, it indicates for each order per user (barring the first order), which items are reorders, and which are first-time orders. They hope to fit machine learning models with this data to provide accurate recommendations for their shoppers in the future.

Approach

The end-goal of the approach to this problem is to build a classifier which takes as input the features of an order (mentioned above), and to predict which items will reappear on the next order. This involves a lot of data processing since the dataset is split over several comma-delimited files, with some missing values which need to be handled intelligently.

The overall number of features in this dataset is not too large (a total of 6 were used in the final model) so we do not anticipate any issues with the curse of dimensionality. However, the dataset is massive, resulting in approximately 60 million rows in the final dataframe after compiling all the constituent components of the dataset. Thus, our main challenge here will be how to deal with this data in such a way that makes it computationally feasible. Furthermore, care must be taken to avoid overfitting to the data and losing the ability to generalize, which is key to any reliable model.

Prior to building our classifiers, we perform some data exploration, visualization, and analysis. This can reveal some hidden trends in the data that may guide our model building to a certain extent. In addition, it will uncover any hidden biases in the dataset and provide us with a more thorough understanding of the cases where our model would perform as expected, and those where it may underperform.

Metrics

The metric used by Instacart for this competition is the F1 score. The F1 score considers both the precision, and the recall. Intuitively, precision is the ability of the classifier not to label negative samples as positive and recall is the ability of the classifier to find all positive samples. Letting t_p denote the number of true positives, f_p denote the number of false positives, f_n denote the number of false negatives, p denote the precision, and r the recall, the formulas to calculate the precision, recall, and F1 score are as follows:

$$p = \frac{t_p}{t_p + f_p}$$

$$r = \frac{t_p}{t_p + f_n}$$

$$F_1 = 2 * \frac{1}{\frac{1}{r} + \frac{1}{p}} = 2 * \frac{p * r}{p + r} \quad \text{(harmonic mean of } p \text{ and } r\text{)}$$

This is a rather appropriate metric to use here since factoring in both precision and recall allows us to see if our model is providing too many false positives (from precision), as well as overall performance (in a more general sense) from the recall which will account for missed positives. It is also slightly harsh since we can see from the formula that, barring the factor of 2, the F1 score must be lower than both the precision and the recall (reminiscent of resistors added in parallel).

Moreover, the scikit-learn library contains a built-in scoring function for F1 making it convenient to use. The maximum attainable value for the F1 score is 1 (a perfect score), and the minimum is 0.

Analysis

Data Exploration

For the data exploration, we are looking to identify any hidden or non-obvious trends in the data that might give us insight into how well or poorly our model might perform in certain test cases. It is also a way of verifying both quality and integrity of the dataset.

We begin by looking at the distribution of our target variable, the reordered feature. Figure 1 below shows the distribution of this variable in a histogram. We see that there is a minor skew towards the reordered side (1) but not too large. More specifically, 59% of the entries are reordered, and 41% are non-reordered, which we are satisfied with.

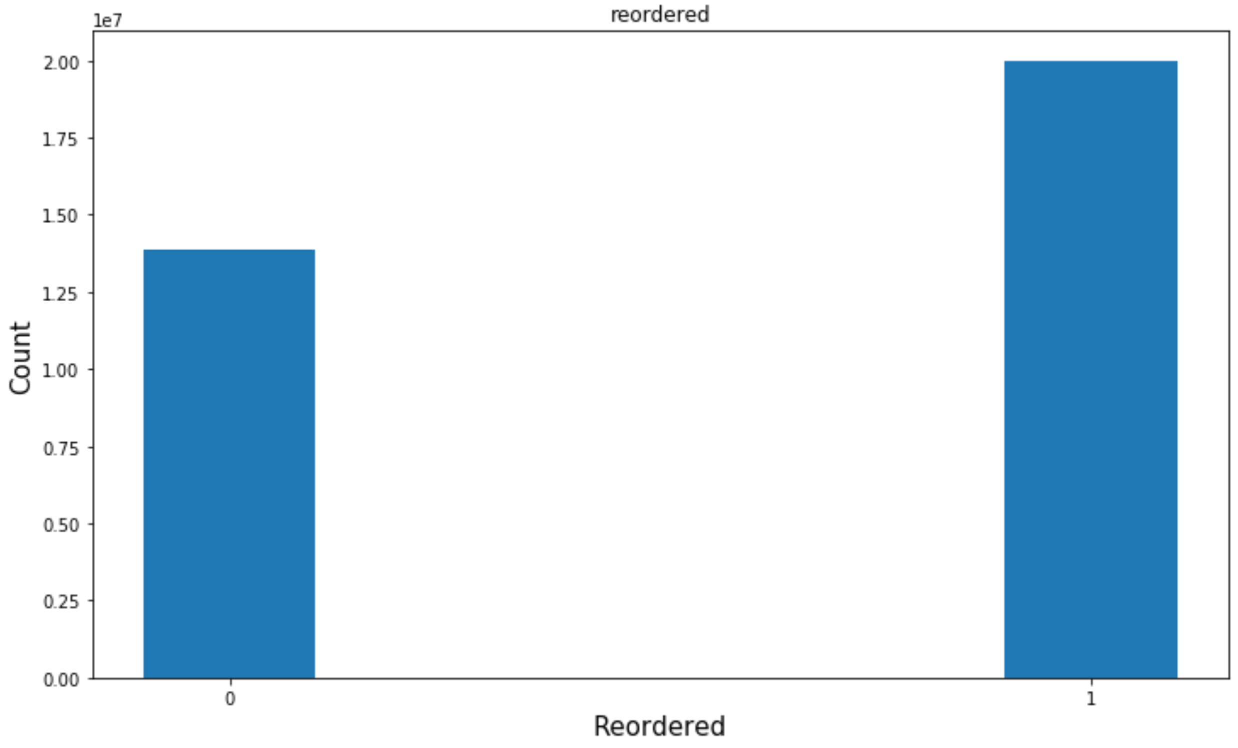

Figure 1 Histogram showing distribution of the Reordered target feature

Next, lets look at the distribution for the order_hour_of_day variable in Figure 2 which indicates the time of day of a given order. We see that the most orders are after hour 6, and plateau between hours 9 and 16. This of course makes sense if we assume that these hours are simply in military format, and most people are asleep between midnight and 6 a.m. The only cause for concern here is that our use of this variable as a model input means we may underperform and lose accuracy for orders that are during off-peak hours.

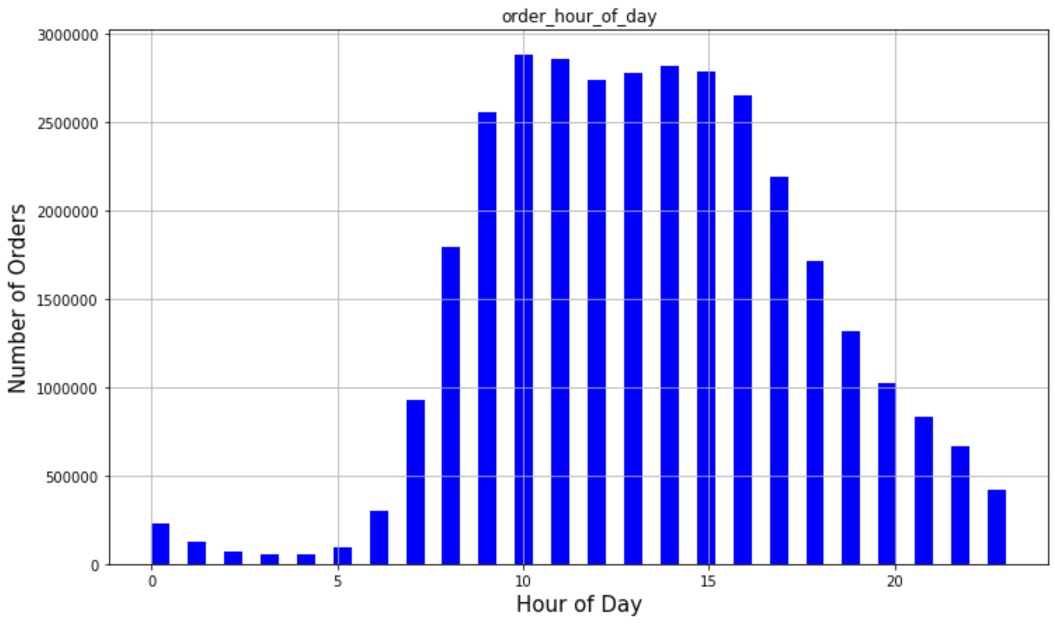

Figure 2 Histogram showing distribution of the order_hour_of_day feature

Now, we observe the same plot but for order_dow (day of week) feature in Figure 3. While there is a trend of decreasing orders from day 0 to day 3, then an increase from day 3 to day 6, the magnitude of this change does not appear to be significant. Specifically, the range is from about 4 million to 6.5 million. Thus, while we do gain some interesting insight from this statistic, there is nothing significant about it in terms of our model.

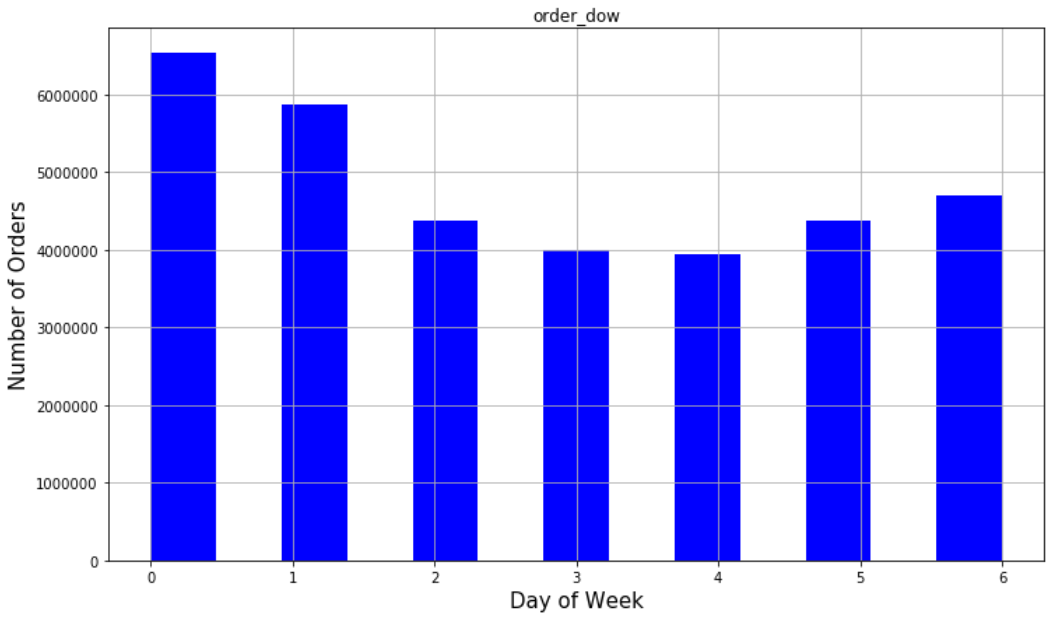

Figure 3 Histogram showing distribution of the order _dow (day of week)

Next we have one of the more interesting statistics. Figure 4 shows the number of reordered items for each “add to cart order”. Intuitively, one would expect that the average shopper starts by adding the essentials to the cart first, and these essentials would typically be the reordered repeated items. This matches exactly what we see in the plot: a monotonically decreasing curve (and a rather sharp one for that matter).

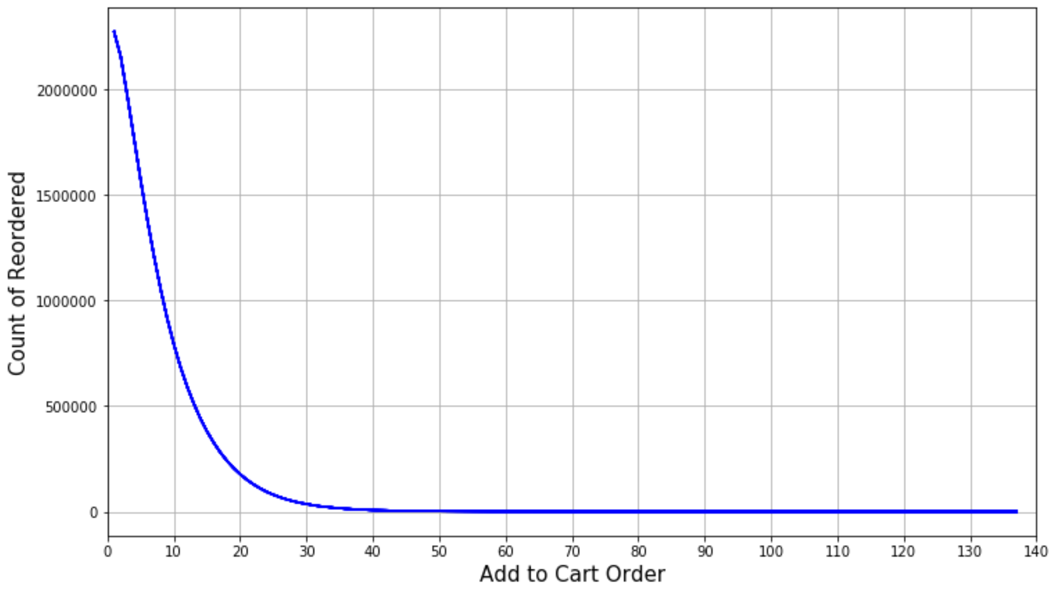

Figure 4 Plot of reordered count vs add to cart order

Algorithms and Techniques

Three different models were used and they showed a ranged of different results:

Gaussian Naïve Bayes (GNB)

Naive Bayes is a supervised learning algorithm based on applying Bayes' theorem with the "naive" assumption of independence between every pair of features [1]. In fact, in our case, this turns out to be a valid assumption since most of the features we feed the model do appear to be unrelated.

Support Vector Classifier (SVC)

Support vector classifiers (SVCs) are supervised learning methods used for classification. Their main advantages are their effectiveness in high dimensional spaces as well as their versatility coming from the kernel functions. Unfortunately, the time-complexity scales as higher than quadratic with number of samples [2]. This is a rather important issue for us given our dataset size. However, we circumvent this issue by limiting the number of samples used to train this model.

K Nearest Neighbors (KNN)

K Nearest Neighbors (KNN) is one of the more intuitive methods (can be used for both supervised and unsupervised cases). The idea is to find a set number of training samples that are nearest in distance (can use different definitions of distance) and use those to classify the test point [3].

Benchmark

Since this problem was taken from a Kaggle competition, a good benchmark for scoring would be the leaderboard scores from Kaggle.

Given that this problem was hosted on Kaggle as a competition, the scores from the leaderboard will be used as a benchmark. But we keep in mind that the scores on the leaderboard are typically much lower than the scores on training data. The top 10% on the leaderboard scored between 0.38 and 0.4. Thus, we should expect about 0.6 or 0.7 for our trained models to account for any overfitting that may have occurred.

Methodology

Data Preprocessing

Most of the time was spent on preprocessing the data. The dataset provided on Kaggle contains 6 comma-delimited files: aisles.csv, departments.csv, orders.csv, products.csv, order_products_prior.csv, and order_products_train.csv. These needed to be imported separately, with care taken to specify the smallest variable type possible for numeric values (int and float) to save memory. Then, they were all merged on the appropriate features. The missing values were then filled, and converted to numeric (using numpy's nan_to_num method). Finally, the prior orders were separated into training features and targets, and the train orders were separated into testing features and targets. We opt not to use any feature engineering since it would be too computationally intensive due to the massive size of the dataset. Moreover, the features appear to be independent from an intuitive perspective so PCA and ICA would not have been very practical.

Implementation & Refinement

The key libraries involved were pandas, numpy, and scikit-learn. We began by training vanilla models with default parameters to get an idea of how well they performed. After this, grid searches were used with 5-fold cross validation to tune some of the model parameters and maximize the score. As mentioned earlier, the massive number of 60 million entries made it very computationally intensive to train some of the models. Of course certain models suffer more than others in this case. The GNB training time was feasible up to about 5 million entries. However, the SVC and KNN models became infeasible for more than 15000 entries. So for these two models, only a subset of 15000 entries was used. Where applicable, random states were predefined to maintain repeatability.

For our gridsearch, the following hyperparameters were tuned:

1. SVC – Mean test scores ranged from 0.6940 to 0.7524:
    a. Shrinking: (True, False)
        i. Whether to use a shrinking heuristic or not
2. KNN – Mean test scores ranged from 0.572 to 0.6105
    a. N_neighbors: (10,11,12,13,14)
        i. Number of neighbors to use
    b. Weights: (uniform, distance)
        i. Weight function to be used in prediction. Uniform implies uniform weights, whereas distance weights points by the inverse of their distance (closer points have larger weights).
    c. P: (1,2,3)
        i. Power parameter for the Minkowski metric. A value of 1 is equivalent to Manhattan distance (L1), 2 is Euclidean distance (L2), and 3 is L3 distance.

Results

Model Evaluation, Validation & Justification

Table 1 below shows the results of the 3 models that were used. It indicates optimal parameters from GridSearchCV, number of training samples, and F1 Score

Table 1 Model Results

Classifier	GridSearchCV Optimal Parameters	Number of Training Samples	F1 Score
Gaussian Naïve Bayes	n/a	5,000,000	0.7489
Support Vector Classifier	Kernel = RBF, Shrinking = True	15,000	0.7537
K Nearest Neighbors	n_neighbors = 13, weights = uniform, p = 1	15,000	0.6114

These results show that the GNB and SVC perform very well. However, GNB used many training samples which means that it could have very likely achieved such a high score because it overfit. On the other hand, the SVC used only a fraction of those samples and achieved the same score, making it much less likely to have overfit, and probably better at generalizing. Another way to view this is that with only 15 thousand samples, the SVC could find a dividing hyperplane, whereas the GNB model took 5 million samples to properly identify differences between reordered and non-reordered products. Therefore, one could conclude that the SVC is simply better suited to this problem than the GNB is, and can detect the patterns within the dataset more effectively.

Thus, we could say that the SVC model appears robust enough for this problem if the dataset size is controlled to keep the computational intensity manageable. It also appears to predict fairly accurately so the results can be trusted. We note that this model may struggle a little bit in cases where the order hour of day is between 0 and 6 since most of our training set did not include data points in this range. But apart from that, it appears the results from the model can be trusted. And finally, its scores appear to be in line with what we expect from the Kaggle leaderboard as mentioned above in the Benchmark section.

Conclusion

Visualization & Reflection

To conclude, let us consider once more the key concern we had above: effect of training subset size on our model's accuracy. Figure 5 below shows, in blue, the F1 score of the SVC when the subset size is swept from 1000 to 15000 (in steps of 1000), and in red, the mean training time (actual time is about 5 times this due to cross-validation).

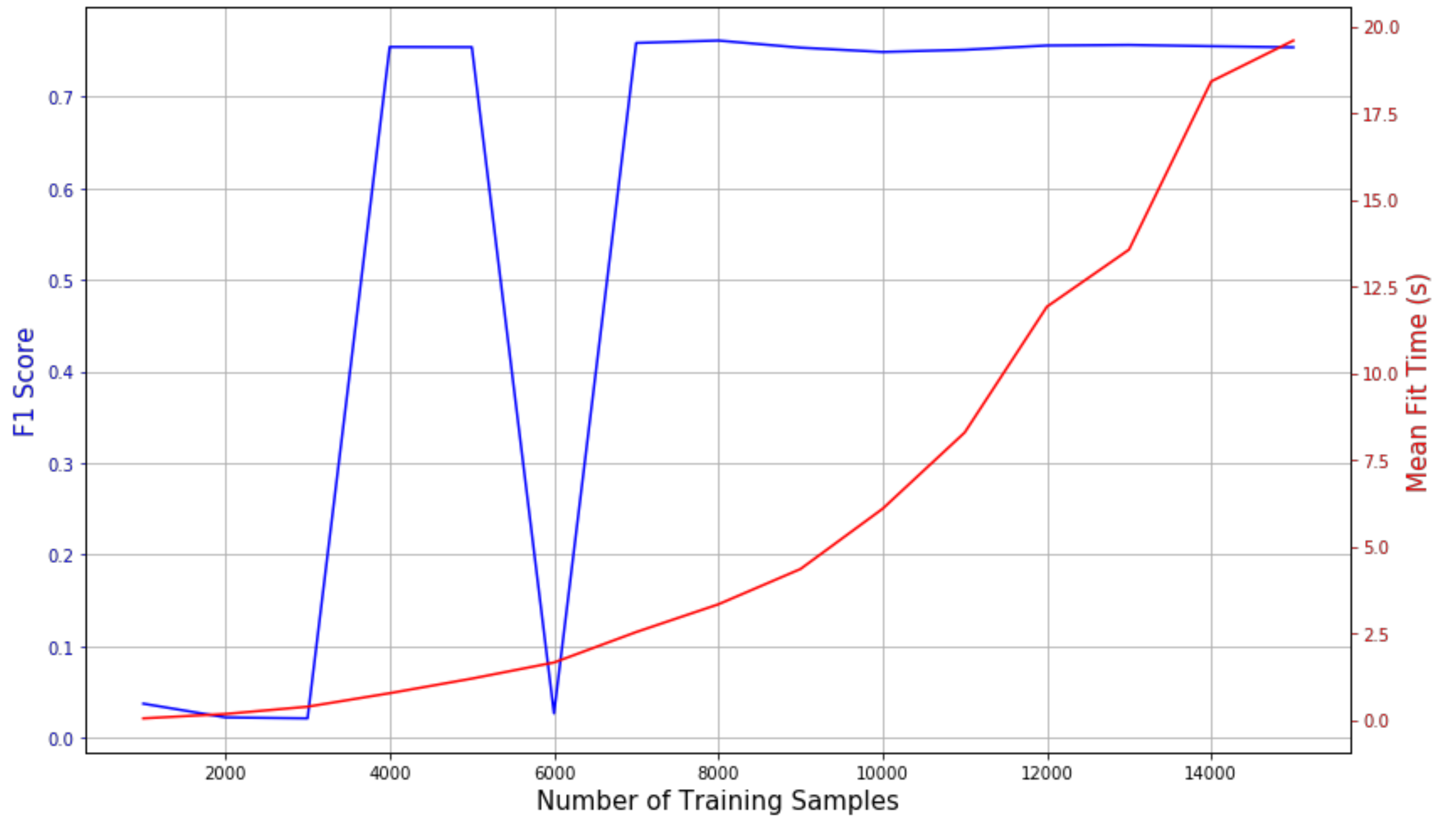

Figure 5 Effect of Training Samples on Score and Training Time

This highlights what we can consider to be a very important aspect of this model: it can learn very well with a very small subset of the data, but if it is used with the full dataset the training times become infeasible. Additionally, it is not recommended to use less than 10000 samples to avoid the oscillatory behavior seen above. Perhaps for a final deployment this would not be the most practical route to take, but for the purposes and scope of our project, it suits us very well. If Instacart were to use this model, a solution would need to be found for the impractical training time it would take on the full dataset. Alternatively, they could intelligently select the subset of the data to make it as broad and representative as possible (as opposed to what we have done, which is select the subset at random).

A final point to recap on was the challenging feat of compiling all the constituents of the dataset into a single comprehensive dataframe. A great deal of care had to be taken to manage memory. This included freeing up memory by deleting intermediate dataframes, as well as manually indicating the variable types as they were imported from the comma-delimited raw files. For example, the hour of day variable had a very small range, so int8 variable type was used. On the other hand, the range for the order number was much larger so int32 was used. Essentially, care had to be taken at every single step to ensure that memory was being accounted for as we progressed through the data preprocessing.

Improvement

The idea of improving our model opens some very interesting avenues. The most intriguing of these would be attempting to use some of the relationships uncovered in the data exploration. For example, we could try the KNN model with a custom distance function. The custom distance function might have some kind of weighting factors on different features of the dataset. Of course, we would require some more in-depth exploration to guide this weighting. Alternatively, we could also just allow the GridSearchCV method to find the optimal weights. Clearly, this is just the tip of the iceberg for the alternatives we have for improvement; but it is in an interesting place to start.

References

[1] http://scikit-learn.org/stable/modules/naive_bayes.html#gaussian-naive-bayes

[2] http://scikit-learn.org/stable/modules/svm.html#svm-classification

[3] http://scikit-learn.org/stable/modules/neighbors.html#classification