Foundation of Machine Learning

Assignment 2 – Kaggle Challenge

Team : The Machina

Sanjana GUPTA Sarvani SAMBARAJU Ayush TANKHA Aline Helburg

Section 1: Feature engineering

In order to develop the best prediction model and to fit our training data, we created multiple features and used engineering techniques. We mainly came up with new features using real-life observations: what does a spam email or a promotion email look like? We first looked at our own inbox: how is an email from Facebook.com, Booking.com, our parents, siblings or a spam structured? Then, once we had analyzed our inbox (which is limited in terms of data and diversity), we read bibliography on the subject. Our main goal was to learn how ubiquitous email providers use machine learning to classify the customer's email. For example, the Google service, Gmail, classifies, by default, the emails in different categories : primary, promotions/ad, social networks, spam, notifications and forums. How such a classification is operated ? Based on our research, the email providers usually base their algorithm on the content of the message: a mail containing the words : viagra, win iPhone or 100,000€ is most likely to be a spam while a mail having the words: booking, reservation number is most likely related to travel. Furthermore, the classification also uses the IP address, the email domain and its specificities (reputation, name, etc.).

We implemented the following features:

1. **Text-to-image ratio**. This ratio was created to capture the importance of text on the images in a given email. This theory was formulated because spammers sometimes display information in large images instead of text so that the filter programs cannot "read" the content. Furthermore, promotional emails usually have a high amount of images while a personal email has low image count. A promotional, travel or forum email has a text-to-image ratio of 80/20
2. **Text-to-link ratio.** Having a high number of links in the body of an email is linked to the label of the email: for obvious reasons, large numbers of links tend to be in promotional emails. But a low text-to-link ratio, meaning that a URL' takes place' in comparison to the number of characters may be linked to a spam email.
3. **Frequency of the emails sent according to the domain and organization**. This feature allows us to show how frequently the user receives an email from the same email address. The frequency was calculated during a period of 7 days.
4. **Hours the message has been sent** (using the time zone). We combined the hours (in the date column) with the time zone given. This feature was created under the assumption that an email sent at night has a higher chance to be an automatic email with the labels: promotions, forums, spam, social network.

To capture the best of the features, we paired the features above with different engineering techniques:

1. **Numerical and Categorical Imputation** : Missing categorical values are replaced by the most commonly occurring value and

missing numerical values are replaced by the mean of the corresponding value in other data.

2. **Categorical encoding** : In order to use the categorical values such as mail type, domain name or the organization, we encoded them using One-hot encoder.
3. **Feature Splitting:** We split, for example, the feature mail type in two.

Section 2: Model tuning and comparison

We tried the following classifiers and pairing them with tuned parameters to better the model's performance and prevent overfitting :

1. **Random Forest**. This classifier was giving us a lower accuracy score on the test set but as well as on the cross-validation than our final classifier (XGBoost). To prevent overfitting using Random Forest, we tuned the following parameters :
   - 1.1. *n_estimators*: the more trees we have, the less likely the algorithm is to overfit.
   - 1.2. *max_features*: we choose a lower number of features (around half of the number of features we initially had). We had to choose a smaller number of features but not too small as it can lead to underfitting.
   - 1.3. *max_depth*: we chose a low number as it reduces the model complexity.
   - *1.4.* *min_samples_leaf:* this parameter was always set larger than 1.

2. **XGBoost**. This classifier was giving us the highest accuracy score on both the test set and on the cross validation. To prevent overfitting using XGBoost, we tuned those parameters:
   - 2.1. *eta (learning rate)*: the value by default is 0.3. We set the value to 1.05 as a higher learning rate makes the computation faster.
   - *2.2.* *max_depth:* the default value is 6, we tried different values between 0 and 6.
   - 2.3. *subsample*: we decreased the value, to be below 1, as it represents the fraction of observations sampled.
   - 2.4. the regularization parameters : *alpha*, *gamma* and *lambda*. Tuning lambda was giving us the best results.
   - 2.5. *n_estimators*: we tried different values from 100 (default value) to 600.
   - *2.6.* *colsample_bytree:* such as the subsample parameter, we decreased the value to be below 1.

3. **Logistic regression.** This classifier did not give us the best result even though we tuned the following parameters:
   - 3.1. *number of features*: decreasing the number of features decreases the risk of overfitting. When using the logistic regression, we limited the number of features to 7.
   - 3.2. *penalty*: either l1(LASSO regression) or l2 (ridge regression). We choose l1, LASSO regression is robust to outliers.
   - 3.3. *C*: we tried different values of C, the strength of the regularization, the values were : 10, 1, 0.1, 0.001.

4. **Nearest neighbors.** This classifier was the 'base model' given as an example. The default value given was k=3. We tuned the following parameters:
   4.1. *n_neighbors*: we tried different values for K but the one that gave us the highest score was k =9.
   4.2. *weight* was set to *distance.*
   4.3. *metric*: we used the Minkowski distance.

5. **Decision tree.**
   5.1. *max_depth*: a low number decreases the model complexity.
   5.2. *min_samples_leaf, max_leaf_nodes* and *min_samples_split* : choosing low numbers but not too low are keys to avoid overfitting.
   5.3. *ccp_alpha*: we picked a ccp_alpha of 0.015, thus the nodes with low relevance are deleted and thus reduces the model complexity.

We used the function *sklearn.model_selection.GridSearchCV* in order to find the optimal hyperparameter values for a given model.

6. **Naïve Bayes.** We chose to discard Naives Bayes classifier as it gave us a bad score on the test set and on the cross-validation on the training data. We used the Gaussian Naive Bayes, Multinomial Naive Bayes and Bernoulli Naive Bayes. They all did not perform well on our dataset. This classifier is not a good choice as it makes an assumption that each attribute is independent of the other attributes - it is not the case. This classifier works also really well with a small dataset. Nonetheless, we test this classifier in order to see what a 'bad' classifier looks like on our dataset.

Table 1. Comparaison of the performance of the models on the training and test dataset

	Cross-validated performance on the training data	Score on the test set
Random Forest	0.5582	0.46386
XGBoost	0.6901	0.53429
Logistic regression	0.5002	0.46386
Nearest neighbors	0.5591	0.46357
Decision tree	0.6582	0.46706
Naives Bayes (Gaussian)	0.2405	0.17547

Edition:

6. CatBoostClassier. Being unsatisfied by our results for most classifiers we decided to use CatBoostClassifier which helps with gradient boost on decision trees. We choose this after a lot of research as it produces cutting-edge outcomes without the substantial data training that other machine learning techniques generally demand and it has strong out-of-the-box support for the more detailed data formats that are associated with many business problems.. We used the 'MultiClass' parameter as the target had different values and set the border_count parameter to None. We tuned the parameters to the following

   6.1. *iterations :* we set this parameter to 25

6.2. *classes_count* : was set as 8 since we have to divide in 8 classes

6.3. eval_metric: the specified metrics for the dataset was set as ‘MultiClass’ since we have 8 classes

6.4. depth : 7

This classifier finally gave us the score on test set to be 69.56% accuracy which when compared to other classifiers gave us the best results

Edition:

Why did CatBoost classifier give better results than XGBoost while both are variants of gradient boosting algorithms?

There are 3 reasons to this:

1. Symmetric trees: CatBoost builds symmetric(balanced) trees, unlike XGBoost. The same condition is used to divide leaves from the preceding tree at each stage. For each of the level's nodes, the feature-split pair that causes the least loss is chosen. The balanced tree architecture reduces prediction time, makes quick model applications, and controls overfitting because the structure acts as regularization. It also helps with efficient CPU implementation.
2. Ordered boosting: Due to an issue known as prediction shift, traditional boosting techniques are prone to overfitting on small/noisy datasets. These methods use the same data instances that were used to build the model, eliminating the possibility of encountering data that hasn't yet been seen. In contrast, CatBoost employs the idea of ordered boosting, a permutation-driven method, to train the model on a portion of data while computing residuals on a different subset, preventing target leakage and overfitting.
3. Native feature support: CatBoost supports any features, whether they are text, category, or numeric, and eliminates the need for preprocessing.

Table 2. Comparaison of the performance of the models on the training and test dataset (edited)

	Cross-validated performance on the training data	Score on the test set
Random Forest	0.5582	0.46386
XGBoost	0.6901	0.53429
Logistic regression	0.5002	0.46386
Nearest neighbors	0.5591	0.46357
Decision tree	0.6582	0.46706
Naives Bayes (Gaussian)	0.2405	0.17547
CatBoostClassifier	0.63	0.6956