A Machine Learning Approach for Customer Churn Prediction in Telecommunication Companies

Introduction

All big and medium sized companies in the globe store data from their current clients, directly or indirectly. This data includes personal data of the client and the interaction that the client does with the services the company provides. While this data is frequently stored and never used again, it can contain a lot of valuable latent information, which can be used to predict the behavior of future customers. Predicting the behavior of a particular current or future customer can be of great interest, since a proactive response to a future necessity or behavior can be given, allowing the company to progress in their one and only objective: make money. Corporations are not dumb though, and recently the ability to extract valuable information from data has been gaining importance in the job market, resulting in an increase of jobs and degrees related to data science.

The goal of this project is to make a predictor, and to construct it from data. In particular, we are taking our data from a dataset called '**Churn**' (1), which is publicly available in the Internet, and has been generated and ceded by an unknown telecommunication company. The predictor will try to predict the value of one variable with respect to the others. In concrete, we want to predict the variable which represents if a client leaved the company ("churned") or decided to stay. This variable can only take two values, and that means that the problem to be solved is a **classical binary classification problem**.

The results to be obtained by the classification model must be the ones that the company which generated this dataset would expect: **the probability of 'churning' (changing of company) of a given client**. This is the most evident and useful information which can be extracted from this data.

Related Work

The techniques that have been used in this report to solve this problem have been discussed and used for years. These techniques are **LDA, QDA, kNN, Naive Bayes, GLM (Logistic Regression), MLP (Sigmoid Output) and Random Forest**. For more information, check (2).

There is not much information about previous work on this data on the Internet. The only results found on previous attempts to perform this specific classification task are found in the **OpenML repository** (3)(4), which is one of the multiple sources for this data. Note that there is no information on data preprocessing or feature extraction/selection, so results may not be accurate.

Note also that the priors for the variable we want to predict are 0.8586 and 0.1414 for staying and churning, respectively. Then, an 85.86% predictive accuracy is to be expected from a model that always decides that the client is staying in the company.

Three methods have been used (among others) to solve this problem:

- **Naive Bayes:** The results obtained with this method are disappointing, the person who tried it was probably not preprocessing and using the method properly. Accuracy: 87.2%
- **Support Vector Classifier:** A type of Support Vector Machine (5) for classification. It has been studied in later stages of the AA1 course. No data on the kernel used is given. Obtains mediocre results considering the priors, but shows progress. Accuracy: 93.5%
- **Random Forest Classifier:** Another method for classification (6), which uses "bagging" in combination with decision trees, explained in later stages of AA1 too. The results obtained are the best along all the methods tried in OpenML. Its results can give an idea of what we can expect from the capability of prediction of our predictive variables. We know that Random Forests are really accurate when we have a high amount of samples, and this result confirms it. Accuracy: 95.5%

Data Exploration and Preprocessing

Basic Exploration and Preprocessing

The churn dataset is in essence a table, like most datasets. It is composed by 21 variables or columns and 5000 columns or entries. As it can already be deducted, each entry represents a client in the telecommunications company. Lets review its raw variables:

- **state:** The American state where the client lives. A categorical variable. The variable takes 51 unique values, representing the 50 states and the District of Columbia. The variable is quite balanced.
  - Changes Made: Converting the variable from integer to factor. Linking the name of the state to the corresponding category.
- **area_code:** A categorical variable, taking 3 values (408, 415, 510). It represents the starting 3 digits of the phone number associated to the client the row represents. It can be useful since it gives a separation between clients. It is balanced.
  - Changes Made: Converting the variable from integer to factor.
- **phone_number:** An integer variable, representing the 4 last digits of the phone number associated to the client the row represents. It is uniformly distributed, as expectable. I have decided to remove it since it does not seem to give relevant information.
  - Changes Made: Removing it.
- **account_length:** It is a continuous variable. The total time the account has been active since the collection of the data. The time magnitude is unknown. It is normally distributed, with mean at 100.

- Changes Made: N/A
- **international_plan:** A categorical (binary) variable, taking values 0 and 1. It indicates if the user has contracted the feature international_plan. It is not balanced, and in concrete only 10% of clients have this feature.
  - Changes Made: Converting the variable from integer to factor. Changing the name of the levels from "0" and "1" to "no" and "yes", respectively.
- **voice_mail_plan:** A categorical (binary) variable, taking values 0 and 1. It indicates if the user has contracted the feature voice_mail_plan. The percentage of clients which have this feature enabled is 26.5%.
  - Changes Made: Converting the variable from integer to factor. Changing the name of the levels from "0" and "1" to "no" and "yes", respectively.
- **number_vmail_messages:** A continuous variable, representing how many voice mail messages the user has sent last month. It is important to note that only users who have voice_mail_plan="yes" can send voice mail messages. The variable is normally distributed with a huge spike in 0.
  - Changes Made: N/A
- **total_ {day,eve,night,intl} _minutes:** A total of 4 integer variables, representing how many minutes the user has spent in calls initiated by the user itself in each time of the day. All four variables are normally distributed, with variable means and variances between them. Note that day=day, eve=evening, night=night and intl=international.
  - Changes Made: N/A
- **total_ {day,eve,night,intl} _charge:** A total of 4 integer variables, representing how many money the company has charged the user for each time of the day. All four variables are normally distributed, with variable means and variances between them. **Very Important**: total_x_minutes is a function total_x_charge (and vice-versa), since the company seems to charge for the amount of minutes spent calling. Per minute prices change depending on time of day and if the call id international. Since it is a linear function of another variable we can remove it.
  - Changes Made: Removing it from the data frame (but keeping it outside for possible future feature extraction).
- **total_ {day,eve,night,intl} _calls:** The amount of calls effectuated by the client for each type of call. Surprisingly, there seems to be no correlation between the amount of calls and the amount of minutes. The variable is normally distributed, with variable mean depending on call type.
  - Changes Made: N/A
- **number_customer_service_calls:** The amount of calls to the customer service effectuated by the user during last month. It is an integer-valued variable, as could be expected. To the naked eye, it seems to follow a Poisson distribution with lambda around 2.
  - Changes Made: N/A
- **class (churned):** Our objective class, representing if the user has or not churned. It is a binary, not balanced variable. In particular, 85.86% of the clients in the database have not churned.
  - Changes Made: Converting it to factor from integer. Changing the name of the levels from "0" and "1" to "no" and "yes", respectively. Renaming it from class to churned.

Note: This database apparently has no missing values, so no removal or imputation of rows has been needed.

Feature Selection

At this point, feature selection is already done, since we have removed the variables which evidently will not help. These are phone_number, total_day_charge, total_eve_charge, total_night_charge and total_intl_charge.

Feature Extraction

The only variable I have decided to extract is the one representing the total money that the client has been charged. It is the sum of (the removed variables) total_day_charge, total_eve_charge, total_night_charge and total_intl_charge. The amount of money the customer is charged can be a determinant factor in the decision of leaving a company. The new variable is called **total_charge**.

It is interesting to note that total_charge is quite normal and that total_charge conditioned to churned = "no" is so, but surprisingly, total_charge conditioned to churned = "yes" is not normal, appearing like a mixture of two normals. This change of distribution depending on the conditioning can be interpreted as a good indicator that total_charge will be a good predictor, since clients tend to churn when the amount of money charged is abnormal (too high or too low).

Shuffling

To end the processing of our data, we shuffle it, to ensure there are no biases. The seed used is 1433.

Resampling Protocol

The 5000 rows are partitioned in two sets, the Training/Validation set, which accounts for 80% of the data (4000 rows) and the Testing set, containing the remaining 20% (1000 rows).

These two sets will be the same for all the methods implemented. In particular, the data is shuffled using the seed 1433 at the preprocessing stage, and from the resulting dataset the first 4000 rows are used for the Training/Validation set and the last 1000 rows for the Testing set. By doing this, we guarantee that all the methods have the exact same data to train/validate/test with, and the comparison becomes more fair.

With every method implemented, the error on the Testing set is computed once all the hyperparameters are set to see how good our **final** model is. The previous statement remains true even if there are not hyperparameters to be set, for consistency.

The exact resampling protocol used to set the hyperparameters and validate the accuracy of our model varies depending on the type of model we are implementing and its computational cost. The data used for this is the 80% located in the Training/Validation set, with no exceptions.

- **LDA: Leave-one-out cross-validation** for the only model possible (no hyperparameters). Accuracy on the test set for the model obtained.
- **QDA: Leave-one-out cross-validation** for the only model possible (no hyperparameters). Accuracy on the test set for the model obtained.
- **kNN: Leave-one-out cross-validation** for every reasonable k. Accuracy on the Testing set for the model with the best k.
- **Naive Bayes: Leave-one-out cross-validation** for the only model possible (no hyperparameters). Accuracy on the test set for the model obtained.
- **GLM: 10 times 100-fold cross-validation** for every set of interactions tested. Accuracy on the test set for the final best models.
- **MLP: 5 times 10-fold cross-validation** for every combination of neurons and regularization tested. Accuracy on the test set for the models which are selected as the best ones.
- **Random Forest: OOB error** on the forest. Accuracy on the test set for a non biased estimation.

  Note: The priors are significantly different for the Training/Validation set and the Test set. (86.25% vs 84.3%, respectively) This can lead to future disparities between cross validation and test accuracy.

Data Modeling

As it has already been exposed, the methods used are be **LDA, QDA, kNN, Naive Bayes, GLM (Logistic Regression), MLP (Sigmoid Output) and Random Forest**. A Support Vector Classifier, as well as other types of classifiers like RBF networks would also work (and probably with good results, as seen in Related Work) on this data, but for time and size reasons those will be omitted.

LDA

LDA has been executed using our **continuous variables**: account_length, number_vmail_messages, total_day_minutes, total_day_calls, total_eve_minutes, total_eve_calls, total_night_minutes, total_night_calls, total_intl_minutes, total_intl_calls, total_charge, number_customer_service_calls.

The results obtained are **horrible**. The estimated (Leave-one-out) Cross Validation accuracy is **86.33%**. Considering that the prior for the class churned="no" of the training/validation set is **86.25%**, it is questionable if there has been any type of learning at all.

This may have been caused because this data is not linearly separable at all. The normality and covariance assumptions which LDA applies on the data are probably too general for the problem in hand.

The accuracy on the test set is **85.1%**. Considering that its prior for 'no' is **84.3%**, the results are better than expected, probably out of pure luck.

QDA

Now it is the turn of QDA, the least restricted version of LDA. The training procedure, as well as the columns used are the same which have been used in LDA. Since we have 4000 rows to train with, the higher amount of parameters that QDA introduces are not a problem.

The results obtained are **mediocre, but there are signs of learning**. The estimated (Leave-one-out) Cross Validation accuracy is **89.18%**, which introduces a significant boost over **86.25%** (prior for churned="no"). It seems that the covariance parameters that QDA includes actually help to create a better separation plane.

The accuracy on the test set is **89.2%**, which is virtually the same error we have obtained with LOOCV (as it could be expected).

Note: Its important to add that both LDA and QDA have been tested with the binary categorical variables of the Churn dataset (converting them to continuous), but the results with those are worse than the results without, and therefore only the results with continuous variables are showcased in this document. The cause may be the poor balancing of the binary classes in combination with the normality assumptions made by the method (which are obviously not true for binary categorical variables).

kNN

Now it is kNN's turn. Even though it does not return probabilities, it is interesting to execute it and see the results. kNN is not suited to work with categorical variables, but binary ones are being included, since in this particular case the concept of order is not present, and should work properly. Then, variables used will be the **categorical ones** plus **international_plan** and **voice_mail_plan**.

The results are obtained by selecting the k which maximizes the accuracy (in the range [1,20]). Leave-one-out cross-validation is being used for each k choice, as it has already been stated.

Non-normalized case

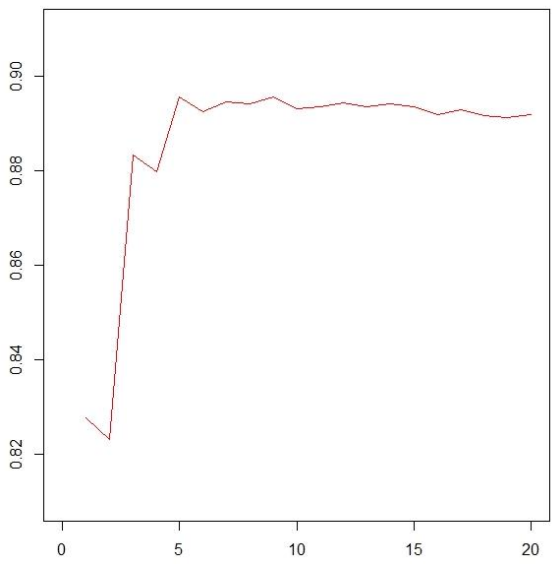
The results obtained in this case are comparable to those of QDA, **89.55%**, with k = {5,9}. There are clear signs of learning, which is good. It starts with a poor accuracy at k = 1 and k = 2 (~82.5%), but at k = 3 (and from it) there is a big reduction in the amount of errors, since the variance starts to reduce and balance with the bias.

Normalized Case

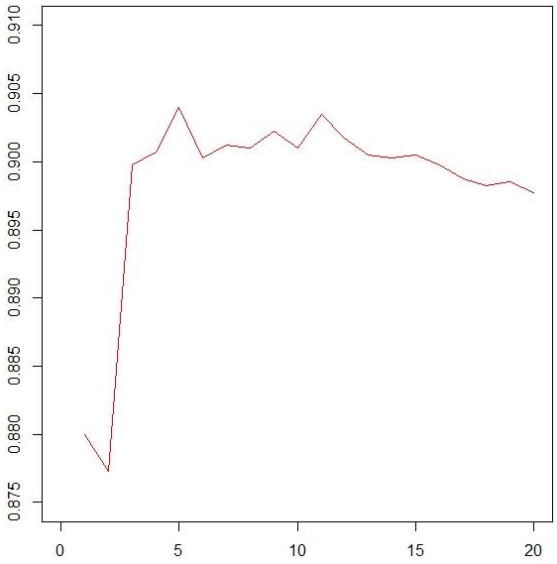
The results here start being interesting. Accuracy is maximized at k = 5, with a **90.4%**. This result is not bad, considering the simplicity of the model. Setting k to 11 also presented good results, with an accuracy of **90.35%**. Surprisingly, normalizing makes the results for low k (1 or 2) much better (82.5% vs 88%).

The normalized case is clearly outperforming the non normalized case because magnitudes matter in KNN and in our data (and in general), a bigger magnitude isn't a synonym of a bigger importance. As an example, our transformed binary variables are not relevant in the non-normalized case, because its tiny magnitude can't compare with those of other continuous variables, like account_length.

From the results above, it can be deducted that k = 7 is probably the best choice for the problem and data in hand, for both the normalized and non-normalized case of the KNN. The worse part of KNN is that to work it needs all the data it has been trained with, which makes it not convenient for a wide range of problems.



non-normalized kNN (accuracy vs k)



normalized kNN (accuracy vs k)

Naive Bayes

In this case, all of our variables are being used, since Naive Bayes is compatible with every type of variable. Note that normalizing the data does not affect the results of Naive Bayes.

Again, leave-one-out cross-validation is being used to validate the accuracy of the model. The accuracy obtained is **89.7%**, which places Naive Bayes slightly above QDA and non-normalized kNN, but below normalized kNN. This implementation outperforms the one found in (4), which probably did a poor preprocessing work.

Looking at the results obtained by all the methods above, we can see a clear tendency to classify churned as "no", since it is the most populated class. This behavior makes the majority of "no" get correctly classified, but at the cost of misclassifying the majority of "yes" as "no". If we tried to fix this by, for example, trying to balance the two classes, the final accuracy on our data with our original priors would be lower, but the predictor would not be biased. For certain applications this unbiased behavior may be preferred.

GLM

In GLM, we finally have hyperparameters (or hyperinteractions, if that makes sense) to be set, so finally the test set is useful and will aid us to see evaluate the accuracy of all the selected models in an unbiased way.

Note that there is a great number of possible interaction variables and an exponential amount of valid models to use in GLM. It can be hard to select the best between-variable interactions. **Multiple anovas (7) and GLMs have been executed in combination with 100-fold Cross Validation to decide which combination to use.** In this report only the most interesting models are being commented.

Naive Model

```
model <- glm(churned ~ state + area_code + account_length + international_plan +
voice_mail_plan + number_vmail_messages + total_day_minutes + total_day_calls +
total_eve_minutes+total_eve_calls + total_night_minutes + total_night_calls +
total_intl_minutes + total_intl_calls + total_charge+number_customer_service_calls,
family=binomial(link=logit))
```

This is the most basic and straight-forward model, including every variable but no interactions. **It has 67 parameters**, principally because state injects 50 additional parameters to the model. The majority of states are not significant, but some of them seem to be significant, according to the anova test.

The **10 times 100-fold Cross Validation** accuracy obtained is **86.7%**. The **test** accuracy is **85.9%**, which is a noticeable drop, probably caused by the prior difference between the training/validation and test sets. This model barely outperforms a model which always chooses to not churn outperforms this model (Which would have 86.25% and 84.3% accuracy on the LOOCV and in the test set, respectively).

Good model, Small

```
model <- glm(churned ~
        voice_mail_plan * international_plan * total_charge +
        number_customer_service_calls * international_plan * total_charge +
        total_intl_minutes * total_intl_calls * international_plan

        , family=binomial(link=logit))
```

This model **has only 18 parameters**. Interactions have been selected carefully, to obtain an efficient, effective model. Due to the low amount of parameters, this model has not overfitted.

The 10 times 100-fold CV accuracy obtained is **92.6%** and the test set accuracy **92%**. The difference, like in the previous case can be explained by the difference in priors. This simple model outperforms any other model that has

been created in this report. I consider it a good result, due to the low complexity of the model.

Good model, Medium

```
model <- glm(churned ~
          voice_mail_plan * international_plan * total_charge +
          number_customer_service_calls * international_plan * total_charge +
          number_customer_service_calls * international_plan * total_intl_minutes *
total_intl_calls

        , family=binomial(link=logit))
```

This model introduces number_customer_service_calls to the last interaction. The **amount of parameters increases by 6, to 24**. Due to the high amount of training samples, overfitting is still not noticeable.

The 100-fold CV accuracy is **92.7%**. The slight increase in complexity translates into a tiny increase in precision, which is good. This model is balanced. The test set accuracy is **92.4%**, which is better than the result of the small one, but this result is within margin of error.

Good model, Big

```
model <- glm(churned ~
voice_mail_plan * international_plan * total_charge +
number_customer_service_calls * total_intl_minutes * total_intl_calls * international_plan *
total_charge

        , family=binomial(link=logit))
```

This model merges the last two interactions of the medium model into one, incrementing the amount of parameters significantly. In particular, the **amount of parameters increases by 12, to 36** with respect to the medium one. At this point, we can start to observe signs of overfitting, but it is not worrying.

The 100-fold accuracy obtained is **93.3%**. This last result starts being competitive, achieving a similar precision of what the OpenML Support Vector Classifier obtained (93.5%) in the OpenML repository. The test set performance is **92.9%**, which follows the same tendency as the obtained in the previous models and does not suggest any abnormal behavior.

MLP

For the MLP models, the library `nnet` in combination with `caret` has been used to obtain the **5 time 10-fold Cross Validation** error in each proposed model. Since we are using the `nnet` library, our feed-forward networks will be limited to one hidden layer. The variable `states` has not been included in the model, since it adds too many input units and the anovas used on GLM suggested that the

error reduction is not worth the complexity added. All continuous variables have been normalized.

To start with, I executed some instances of the MLP with 10 hidden units, heuristically finding a reasonable range for the regularization parameter. Once I found the orders of magnitude where the model seemed to give the best results (10^-2 to 10^0), I ran with `caret` multiple 10 hidden unit MLP with regularization values in that range, to find the best one. As it has been commented, on each regularization value candidate, 5 times 10-fold Cross Validation has been executed.

The table generated with `caret` is:

	size	decay	Accuracy	Kappa	AccuracySD	KappaSD
1	10	0.01000	0.9293	0.6836	0.01326	0.06324
2	10	0.01585	0.9333	0.7028	0.01414	0.06348
3	10	0.02512	0.9375	0.7177	0.01391	0.06770
4	10	0.03981	0.9409	0.7333	0.01155	0.05496
5	10	0.06310	0.9385	0.7214	0.01271	0.06061
6	10	0.10000	0.9385	0.7202	0.01423	0.06667
7	10	0.15849	0.9401	0.7259	0.01274	0.06124
8	10	0.25119	0.9395	0.7222	0.01303	0.06368
9	10	0.39811	0.9381	0.7131	0.01215	0.05995
10	10	0.63096	0.9346	0.6907	0.01052	0.05364
11	10	1.00000	0.9314	0.6672	0.01109	0.05812

Looking at it, it seems that a good range for our regularization parameter is between 0.04 and 0.16. In particular, a decay of ~0.04 is the one which offered the best performance in this particular case, but there is margin of error. The accuracy obtained is **94.1%**, which is a really good result which lands above our best GLM (93.3%) but still below the Random Forest Classifier found in OpenML (95.5%).

The model trained with seed 1433 and the optimal weight decay value obtains a **93.6%** test accuracy, which may seem a bit underwhelming. Running it several more times confirms that the mean test accuracy is indeed a little higher, around 94-94.5%.

For the sake of science, I have repeated the process above with different numbers of hidden units, concretely 20 and 7.

For the 20 hidden unit model, it is noticeable that the **optimal decay value has increased** significantly, about an order of magnitude, from ~0.04 all the way up to ~0.5. The CV accuracy with this more complex model is **93.5%**, which for some reason is below the 10 hidden unit one (94.1%).

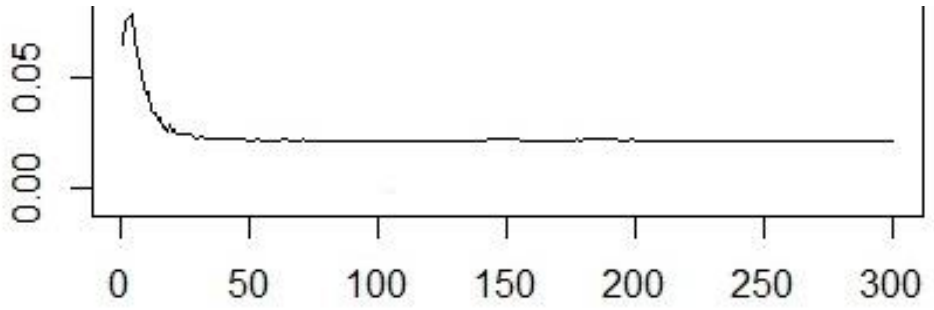
Now examining at the **7 hidden unit** model, we can see that the **optimal decay value is very similar to the 10 units one**. The new optimal value is ~0.056. The cross validation accuracy with 0.056234 decay is **94.5%**, which is **even a better result** than the one obtained with the 10 unit one. The test accuracy of the 7 hidden unit model with 0.056 decay and seed 1433 is **94.9%**, which is good.

Note that the "small" 7 hidden unit model contains a total of 127 weights, which is considerably more than what the GLM models had.

Random Forest

For space reasons, this learning algorithm on this problem will be discussed briefly, but I thought it would be worth to test how good this method behaves with our preprocessed data given the OpenML results (95.5%).

The model has been trained with 300 trees (and seed 1433), which is an "overkill" for most cases but since we have the computational power we can afford it. The error seems to plateau at about 50 trees (and even before).



Random Forest (OOB error vs # of trees)

The results are absolutely outstanding, the OOB error is 2.1%, which translates into a **97.9%** accuracy. The OOB error is usually pessimistic (has bias), and it still appears to clearly outperform any model reviewed in this report. The test accuracy is also **97.9%**, which is the same as the OOB and confirms that **Random Forests work incredibly well with the Churn dataset**.

```
      Pred
Truth  no yes
  no  842   1
  yes  20 137
```

Taking a look at the confusion matrix for the test set predictions, there still is a tendency to classify some examples as false-negatives, which is caused by the prior probability of churned. With this degree of accuracy, it is not a big problem. Note that "no" get practically always classified as "no".

The model with only 50 trees seems to have a similar performance, so it may be preferred over the 300 one. It provides a faster training and evaluation time, as well as reduced complexity with virtually the same performance. In particular, its OOB error is 2.2% (97.8% accuracy) and the test accuracy 97.7%.

The Final Model

In this project, each method tested accidentally offered better performance than the previous one, from the infamous LDA all the way to the surprisingly good Random Forest.

It would be easy to say that the Random Forest is the best model but we will consider each model one by one, **from the worst to the best** ones (in my opinion after seeing the results).

- **LDA:** The data is not appropriate for this method. It is not even worth to consider it as a good candidate, since it didn't show any learning at all.
- **QDA:** Showed progress with respect to LDA, at the expense of more parameters. Nothing remarkable.
- **Naive Bayes:** This method performance is similar to that of QDA, but with a lower complexity.
- **kNN:** In the normalized case and for a carefully selected k, it showed good results. It is too complex since requires all the data (4000/5000 rows) to work.
- **GLM:** The model with only 18 parameters is really precise for its complexity. It can be a good option for **the user who wants to understand the model**, since its linearity and simplicity enables it.
- **MLP:** A great accuracy, at the expense of a great complexity. Probably **the first model that a newbie data scientist would try**. With `nnet` the setup could not be easier, with just a statement to normalize and another to train the model. The estimation of exact hyperparameters takes time, a software which could use more than 1 core and even the GPU would be useful.
- **Random Forest:** I am not able to explain why this method works so well with this data, but **to obtain the best prediction accuracy, a Random Forest is the way to go**. With 50 trees, the results obtained are almost as good as they can get.

Then, I would propose 3 final models:

- **GLM:** The `Good model, Small`, with a 10 times 100-fold CV accuracy of **92.6%** and an accuracy on the test set of **92%**.
- **MLP:** A feed-forward 3 layer neural network, with 7 hidden units and 0.056 regularization factor. The accuracy is **94.5%** and **94.9%**, for the 5 times 10-fold CV and test set, respectively.
- **Random Forest:** A random forest with 50 trees. The OOB accuracy is **97.8%** and the test one is **97.7%**.

Conclusions

The most valuable lesson that this project has taught me is that **there is no thing as the "Master Algorithm"**, an algorithm that outperforms every other method and will learn any type of data and pattern. Each dataset will have an algorithm that fits it correctly (given a good preprocessing). In this case a **Random Forest is an algorithm which fits really well the distribution of `churn`**. I am still having a hard time believing that a concatenation of simple trees, which would be understood even by a child can outperform the feed-forward neural network (which is taught as the end-all learning method in a multitude of "expert" online deep learning courses).

I also noticed that having a deep **understanding of the learning algorithm and the data we are using** is as (if not much (MUCH) more) important as obtaining good empirical results. LDA might not seem even worth trying for this data, but testing it and understanding the nature of the algorithm can teach you something about the dataset in hand which can indirectly help you build a better model for this and future data. I have to admit that I still am a bit "obsessed" about performance metrics, but it is getting fixed over time.

Since R was introduced to me at university a year and a half ago, I was skeptical about its usefulness as a long term Python user. The project and AA1 lab sessions, though, has made me look at R differently, and I am sure I will use it as a tool for my future Deep Learning projects, since it provides **fast prototyping of algorithms and very diverse plotting capabilities**. That does not mean I will abandon Python and PyTorch, but that all this new R knowledge will be a major addition to my toolbox.

Future Work

- There is work to be done in the `state` categorical variable, in concrete a dimensionality reduction one. While it has been shown that for some States the differences in the `churned` variable are significant using anova tests, in practice we have shown that it is not worth to include the amount of extra parameters which this variable introduces for the negligible gain in accuracy.

- It would be convenient to test how this data behaves with other algorithms studied in this course, like Support Vector Classifiers or RBF networks. As it can be seen in the OpenML repository (3)(4) those and other methods obtain good results, and probably can provide valuable information about the data in hand.

References

[1] **OpenML churn** https://www.openml.org/d/40701

[2] **Statistical classification - Wikipedia** https://en.wikipedia.org/wiki/Statistical_classification

[3] **OpenML Supervised Classification on churn** (1) https://www.openml.org/t/167141

[4] **OpenML Supervised Classification on churn** (2) https://www.openml.org/t/146227

[5] **Support-vector Machine - Wikipedia** https://en.wikipedia.org/wiki/Support-vector_machine

[6] **Random forest - Wikipedia** https://es.wikipedia.org/wiki/Random_forest

[7] **Analysis of variance - Wikipedia** https://en.wikipedia.org/wiki/Analysis_of_variance