Turnover Classification Challenge

Richemont

Dataset
The file dataset.csv contains anonymized employees related data.
Every employee has:
- An id: employee_id
- A set of features: f1 to f13
- has_left: it take 1 as a value if the employee left and 0 if he is active today

Goal
The goal is finding a solution to anticipate employees leaving.

Question
1. Build a model able to predict if an employee will leave soon
2. Describe all the steps you did in order to build the model
3. Explain how you evaluated your model
4. Ship in a zip file the code you developed with a documentation about:
   a. How to train a model using your solution
   b. How to test it with new data
   c. An imagination about how to deploy the solution you developed is production

Steps to build model

Exploratory Data Analysis

After ingesting the data, the first step I took in order to build the model was to do exploratory data analysis of the dataset.

Missing Data

The first step of this exploratory analysis was to look for anomalies in the data such as missing data (null values). For most machine learning models, data with null values cannot be used for training/fitting the model. It is therefore important to understand which features have null values and to either drop the rows with null values from the training data, or to impute (fill in) the null values.

Ideally we do not want to drop any rows because the model could potentially miss out on key information from the other non-null values in that row. However, imputation strategies, such as filling the null values with the average or most frequent value in that feature, require a good understanding of the data and the context, so should not be done lightly.

In the case of this dataset, only the feature “f5” contains null values. “f5” contains 21 null values and because they only account for 0.5% of the values in the feature, the safest option is just to drop these rows from the training dataset. A visual inspect of these 21 null records shows that there is no clear commonalty between them, e.g. the “f2” feature does not contain only “not assigned” values, therefore it can be assumed that these 21 records have nulls due to data quality issues.

Analysis of Target Variable

The target variable is the feature that we are interested in predicting, in this case the feature “has left”. The values for this variable are 0 and 1, which I will refer to as not turnover and turnover respectively.

An analysis of the target variable shows that these dataset contains a slight imbalance towards not turnover. 42% of “has left” values are turnover and 58% of its values are not turnover. Imbalances of the target variable can cause issues for classification machine learning models where the model has an incentive to classify new observations as the majority class because the probability of an instance belonging to the majority class is higher.

The imbalance in this dataset is not particularly large, but because in real turnover datasets the imbalance would be significant unless it is addressed it will affect the performance of the model.

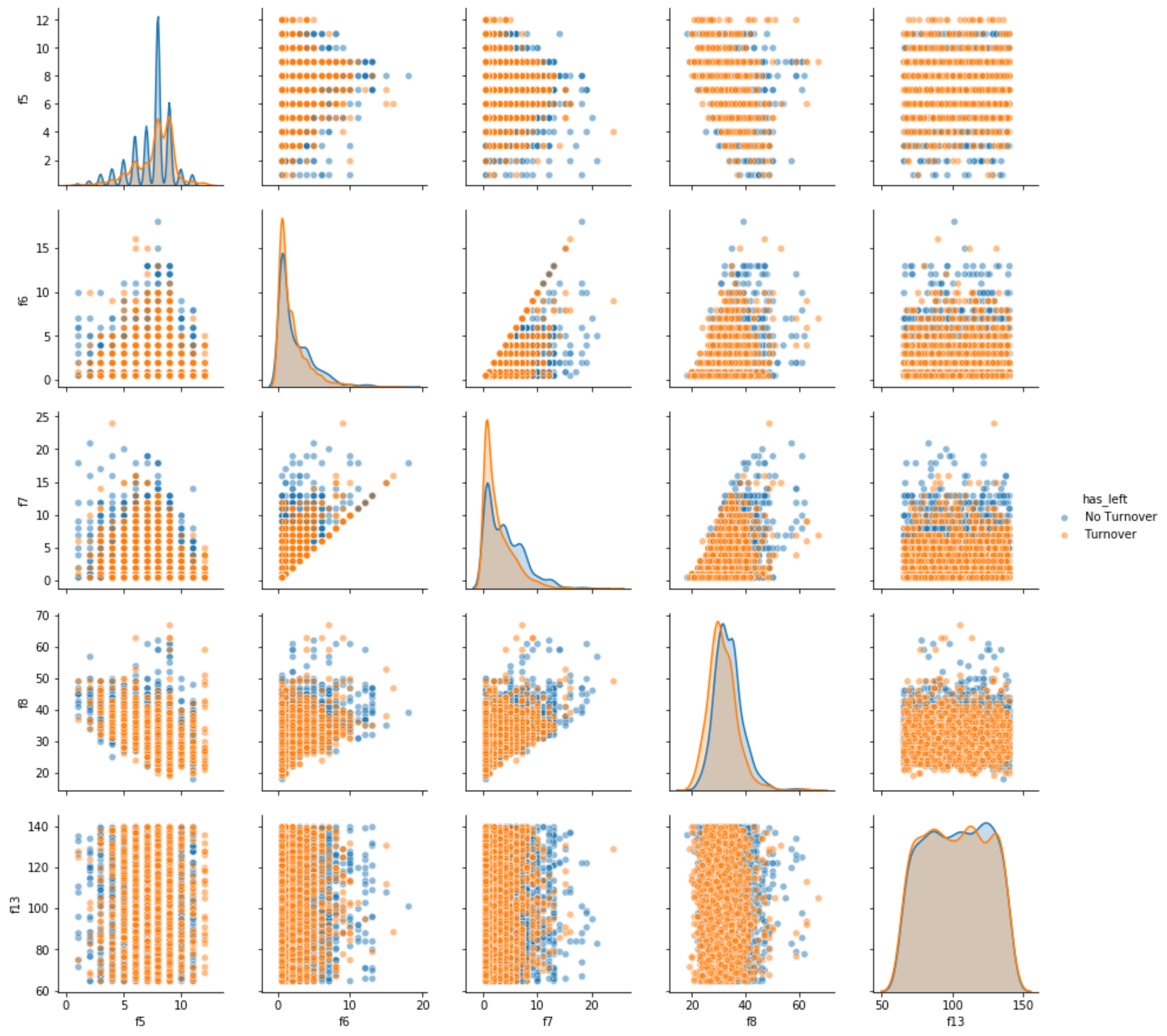
Analysis of Predictor Variables

Numeric Variables

The features employee_id, f5, f6, f7, f8, f11 and f13 all contain numeric data which can be ingested directly into a machine learning model.

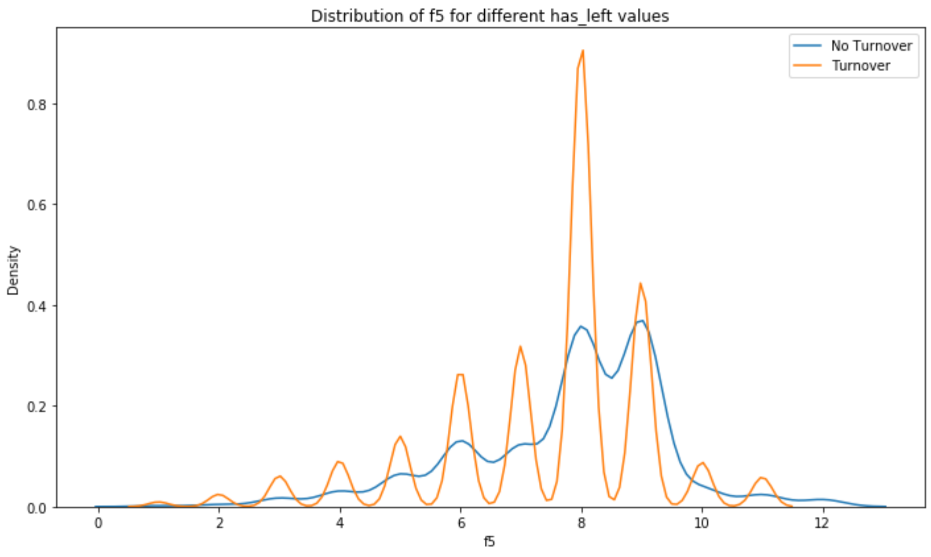
Employee_id however is the unique primary key identifier assigned to each employee and as such needs to be filtered out of the data before it is used by a machine learning model. Inspection of this feature shows no obvious data quality issues such as duplicates.
f7 initially showed to be a categorical variable, but inspection of its values showed that there is a data quality issue where one of it’s values was a “#”. Removing this record turns it into a numeric feature.

The plot below is a pair plot showing the distribution of each of the numeric features as well scatter plots showing the relationship between each of the numeric variables. It is coloured by the target variable, has_left, with turnover in blue and not turnover in orange.

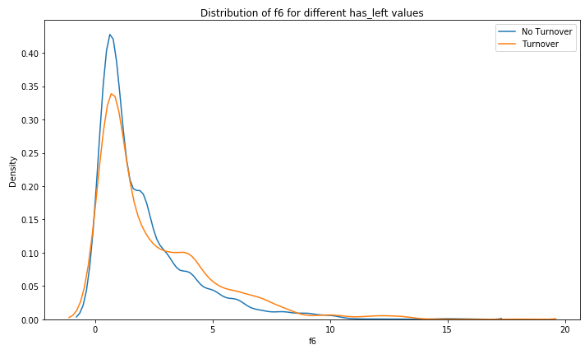


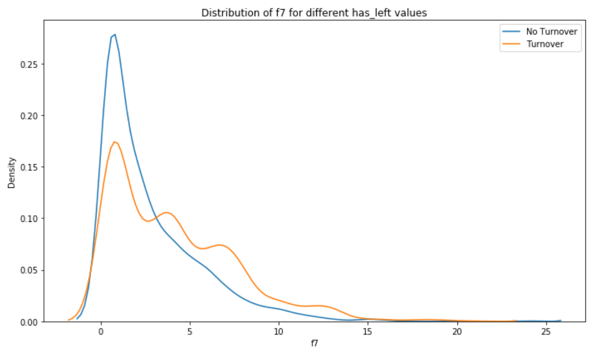
The following plots show the distribution of each feature for the different values of the target feature, has_left.

As seen below, f5 has the highest density when its values are around 8.

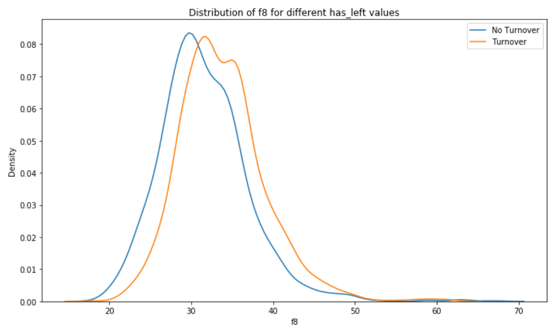


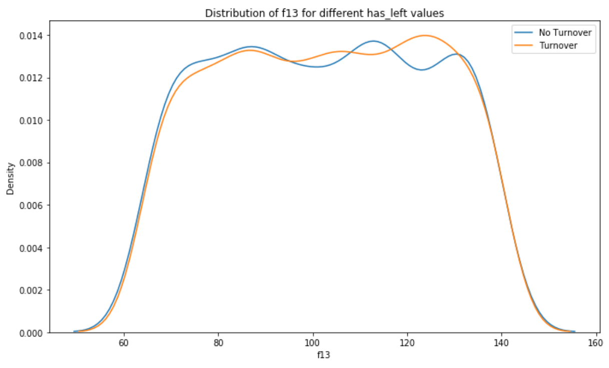
The plots below show f6 and f7. Both of these features have similar distributions which are right skewed having a higher density of lower values.





The plots below show f8 and f13. f8 is close to normally distributed around a mean of 30, with turnover having a slightly higher mean than not turnover. f13 has a consistently high density.



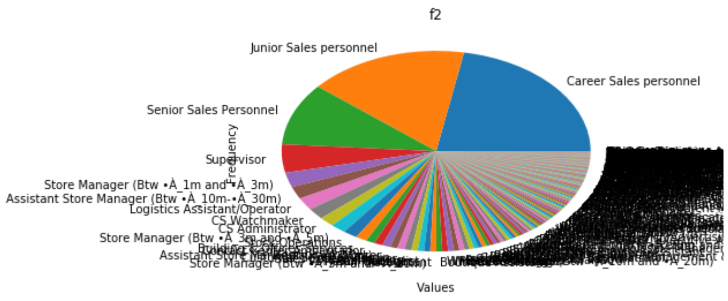


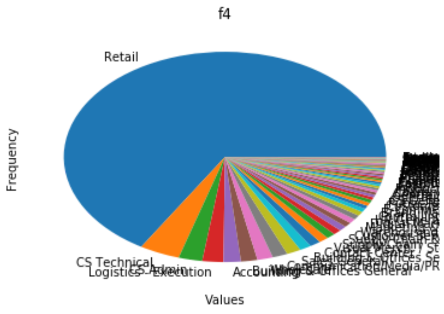
Categorical Variables

The categorical, or nominal, features can broadly be categorised into three types.

1. f2 and f4 are features with high cardinality, meaning they have lots of potential values.
2. f1, f3, f10 and f12 have a manageable number of potential values (<25)
3. f9 and f11 have very few potential values (<4)

f2 contains 216 potential values and f4 contains 67 potential values. The pie charts below show the makeup of these two features.



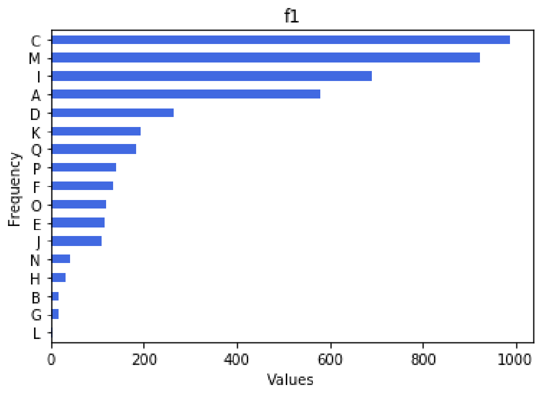


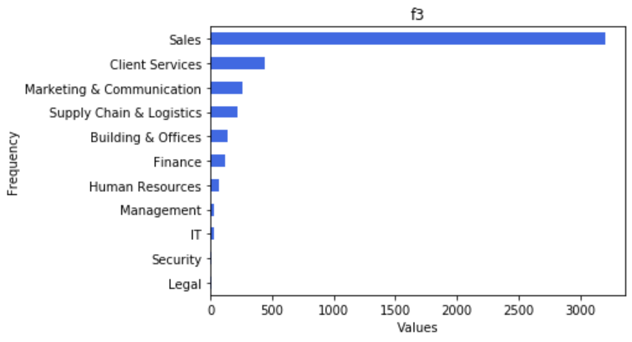
The high cardinality of these features presents a potential problem for any linear model we train. The typical approach to dealing with these features is to one-hot encode them (explained and discussed in the “Preprocessing” section below) but with so many potential values, this approach will result in creating the so called “curse of dimensionality”. For tree-based models, such as Random Forest, these two features can be label encoded (explained and discussed in the “Preprocessing” section below) during preprocessing and included in training data.

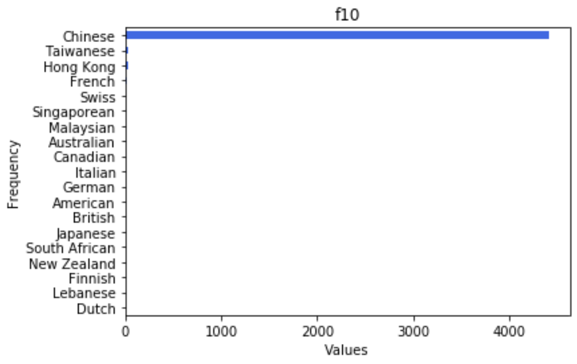
Another potential preprocessing route not taken in this analysis due to time constraints would be to group the least frequent values together to become a new value, “other”.

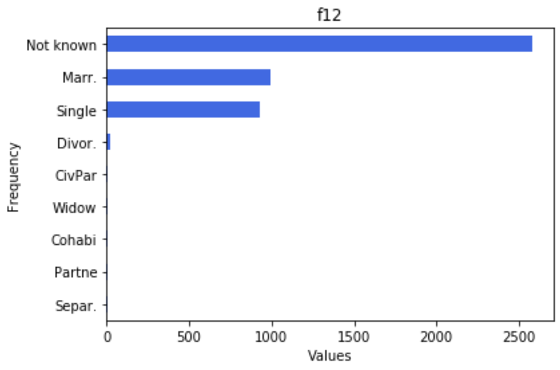
Another approach could be to use business understanding to group similar values together. For example, in f2, from a business perspective there may be no difference between a Senior Sales Personnel and Career Sales Personnel it could just be that in different subsections of the business refer to the same role with different titles. Also, this data was not inspected for data issues such as typos and duplicates which could be creating more classes. Cleaning the data of these issues could reduce the cardinality of these features.

Bar charts for f1, f3, f10 and f12 are shown below. As can be seen each of these features has a much smaller number of potential values than the features above. In order to include these features in modelling, they could be one-hot encoded or label encoded.

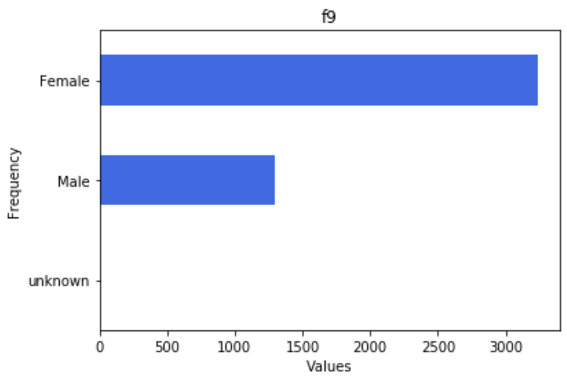


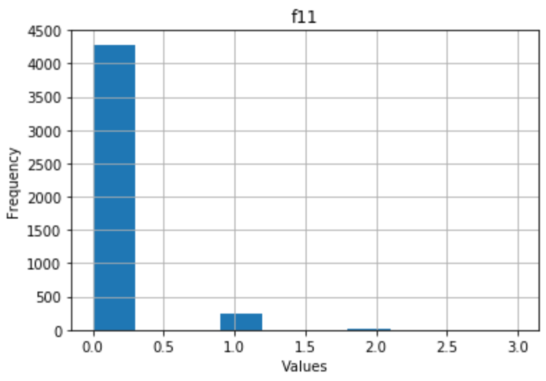






As can be seen below, f9 and f11 have very few potential values. For f9 in particular, the value “unknown” occurs in only one record and as such this was excluded from modelling.





Preprocessing

In order to prepare the data for modelling I experimented with each of the following preprocessing and feature engineering techniques:

- Train test split
- Cross Validation
- Resampling
- PCA
- One-hot encoding
- Label encoding
- Polynomial feature engineering
- Subtracting features

These are each discussed in the appendix.

Models

I considered 3 models for this task,1 linear and 2 nonlinear models. The linear model was Logistic Regression (LR) and the two nonlinear models were Random Forest (RF) and XGBoost (XG). Each of these is discussed in more depth in the appendix.

How I evaluated the model

Evaluation Metrics

I considered the following metrics as well as examining the confusion matrix when evaluating the models (each are discussed in the appendix):

1. Accuracy
2. Precision
3. Recall
4. Average Precision

Examining only accuracy can give a false view of model performance. This is due to the accuracy paradox which states that models with lower accuracy can have better predictive power than models with higher accuracy. A more balanced way to evaluate model performance is to look at not just accuracy, but also other metrics such as precision, recall and Average Precision.

Precision is the quality of predictions based on positive predictions, regardless of all it might miss and Recall is the ability of correctly classify those at risk of turnover.

With a high Precision score, if the model identifies an employee as likely to turnover then we should take it very seriously because it nearly always predicts that someone will turnover when they do actually turnover. However, if the model has a low recall score, the model will not do a good job of correctly identifying employees who are at risk risk of turnover.

Between recall and precision, I would recommend prioritising recall because due to the tangible and intangible costs of turnover, it is better to incorrectly target employees as at risk to turnover than miss out on those who will turnover.

In the context of imbalanced target variables, measuring the Average Precision score is a good way to evaluate model performance. Average Precision summarizes a precision-recall curve as the weighted mean of precisions achieved at each threshold, with the increase in recall from the previous threshold used as the weight.

Baseline

In order to properly evaluate the impact of different preprocessing and modelling options, I created a baseline to be used as a benchmark against which future experiments could be compared.

For this baseline, I trained three models (Logistic Regression, Random Forest and XGBoost) using only the numeric type data from the dataset. The features used were f5, f6, f7, f8, f11 and f13.

The results of the baseline experiment when the models were used to predict values in the test data were follows:
Logistic Regression
- Average Precision 0.595
- Recall 0.378
- Precision 0.624
- Accuracy 0.6197

Random Forest
- Average Precision 0.571
- Recall 0.429
- Precision 0.590
- Accuracy 0.6110

XGBoost
- Average Precision 0.634
- Recall 0.418
- Precision 0.667
- Accuracy 0.6465

Results of different preprocessing and modelling experiments

One-hot encoding

One hot-encoding the categorical features increased the Average Precision of the three models
LR Average Precision = 0.735
RF Average Precision = 0.698
XG Average Precision = 0.747

There is the potential that refining which one-hot encoded columns are included in the training data could increase the predictive performance of the models but due to time constraints this was not explored.

Label encoding

Label encoding the categorical variables instead of one-hot encoding them resulted in the following modelling performances
LR Average Precision = 0.582
RF Average Precision = 0.764
XG Average Precision = 0.784

Unsurprisingly label encoding does not work with a linear model as it interprets the assigned labels to contain linear relationships, i.e. it would interpret a label of 4 as twice as big as a label of 2 when in fact they are arbitrarily assigned integers assigned to particular categorical values.

For the two tree-based models, label encoding resulted in better performances than one-hot encoding. This is likely due to one-hot encoding increasing the dimensionality of the training data to too high a level relative to the number of samples in the training data.

Resampling

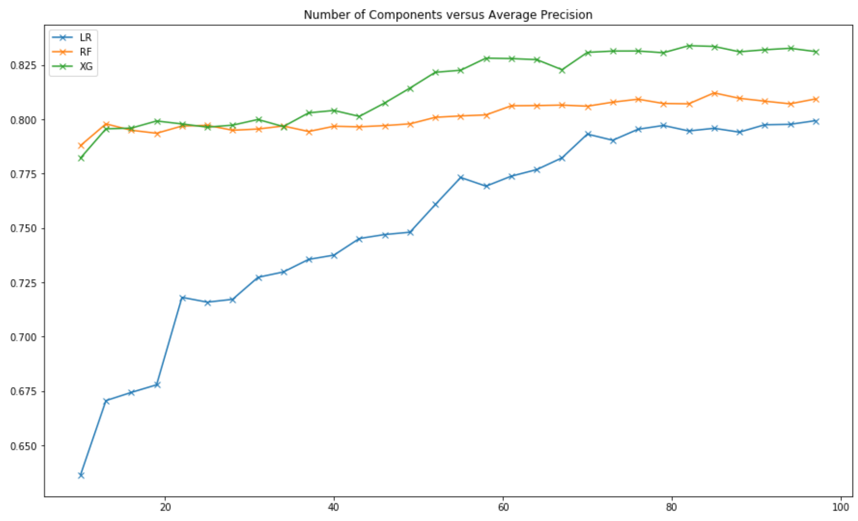
Resampling did not overly effect the average precision of the tree-based models but it did have a positive effect on their recall scores. Compared to label encoding with resampling increase recall scores from 0.544 to 0.677 for XGBoost and from 0.577 to 0.617 for Random Forest.

PCA

I experimented with first one-hot encoding all of the categorical variables, and then performing principal component analysis on those one-hot encoded features.

The plot below shows the relationship between Average Precision and the number of principal components used. Particularly for the linear model, this greatly increased the

Average Precision score.



The Average Precision scores when 60 principal components was used are shown below.
LR Average Precision = 0.784
RF Average Precision = 0.797
XG Average Precision = 0.832

The results are better for all three algorithms than when one-hot encoding was used and when label encoding was used.

However, using principal components reduces the interpretability of the models, particularly when feature importances are calculated and when inferences are calculated for the linear model. Without the ability to make insightful inferences, the benefits of using a linear model over a nonlinear model are reduced. Therefore, for the purposes of this report I would recommend using label encoding as a data preparation step rather than one-hot encoding mixed with principal component analysis. However, in the future, the potential performance gains from this approach should be tested.

Polynomial Features

Creating polynomial features is an automated form of feature engineering in which new features are made by multiplying existing numeric features together, including multiplication by themselves. Training models with these engineered features did not improve model performance.

Subtracting features

Subtracting features is as simple as subtracting one numeric feature from another. This linear transformation while simple, often results in large performance gains in nonlinear models such as XGBoost. For this problem set however, subtracting each feature from every other feature did not boost model performance.

With both the polynomial features and the subtracting features approaches, there could be increases to model performances if feature selection techniques were applied to include only the most predictive features for modelling. However, this was not explored due to time constraints.

Hyperparameter Tuning

The several XGBoost hyperparameters were tuned using randomised search with cross validation. As opposed to a grid search, which trains a model for every possible combination of selected hyperparameters, randomised search trains models with randomly combined combinations of hyperparameters. This randomised approach is commonly a more effective method of hyperparameter tuning than a grid search.

The data used for this tuning was the label encoded categorical features along with the numeric features. The hyperparameters selected for optimisation were

- ‘max_depth’
- ‘learning_rate'
- 'subsample’
- 'colsample_bytree’
- 'colsample_bylevel’
- 'min_child_weight'
- 'gamma'
- 'reg_lambda'

The results of this tuning were that randomised search’s best estimator did not outperform the default XGBoost hyperparameter settings.

Number of Estimators

One hyperparameter which did increase the performance of the XGBoost algorithm was ‘n_estimators’.

With the label encoded categorical features along with the numeric features used as input data, increasing the number of estimators from 100 to 10,000 resulted in the Average Precision increasing from 0.791 to 0.833.

Recommended model

Given the results above, the model I would recommend for further investigation is an XGBoost classifier with default hyperparameters, except for n_estimators which should be set to at least 10,000, and data preprocessing that includes the numeric features along with label encoded categorical features.

The reason for recommending this model is that it produces the best performance for Average Precision and recall whilst also being interpretable, in terms of having visibility into the feature importances.

Feature Importance

After a linear or tree-based model is trained, it is possible to extract from it how important each feature was for influencing its predictions (feature importances are discussed further in the appendix).

The XGBoost model had the following feature importances:

	Feature	Importance
1	f13	0.1805
2	f2	0.1619
3	f8	0.1485
4	f7	0.1065
5	f4	0.097
6	f1	0.092
7	f6	0.0628
8	f5	0.0482
9	f12	0.0304
10	f3	0.0277
11	f9	0.0154
12	f11	0.0149
13	f10	0.0141

As can be seen, f13 had the highest importance followed by f2 and f8. f10, f11 and f9 had the lowest importance.

How to train a model using my solution

I created a separate train.py file which can be used to train a model using my solution. It imports and preprocesses the data file and fits an XGBoost model to it. The program then outputs this trained model along with evaluation metrics, such as the confusion matrix.

How to test the model with new data

In order to test the model with new data I created a separate test.py file which accepts the location of a trained model along with the location of a testing data file. It then outputs evaluation metrics about this new testing data.

An imagination about how to deploy the solution you developed in production

Local vs Server Deployment

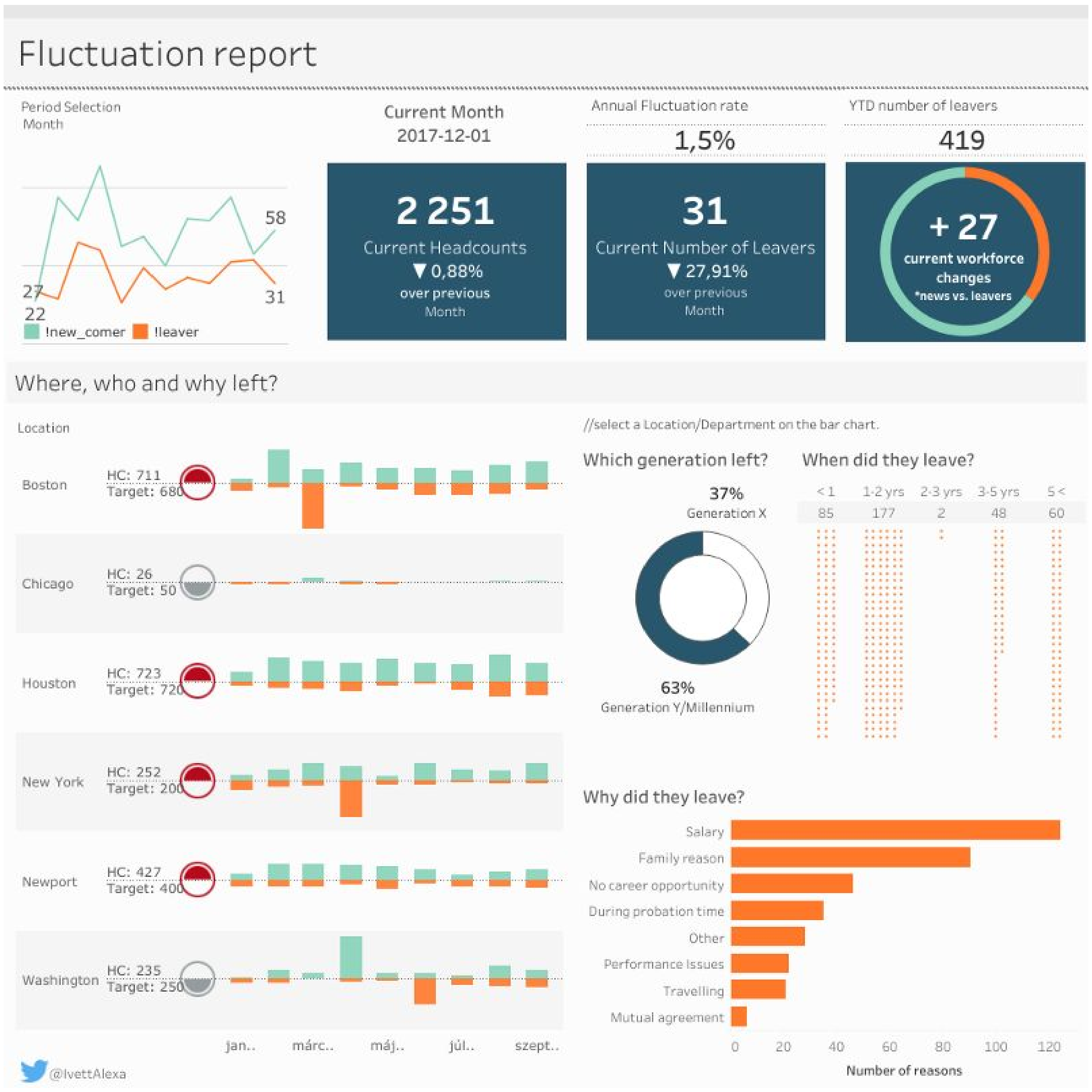
The solution I developed can be deployed in a number of ways. The model can either be hosted on a server or stored locally.

A local deployment will struggle with large datasets in terms of storage requirements and processing times; it has to be trained manually; and it will not produce automated predictions for integration with dashboards or reporting services. However, local deployment can be an option if the following is true: the dataset is small; live updates are not required; the data is very messy and requires careful manual cleaning before being fed into the model; or manual validation of predictions is required by a data scientist and/or HR business expert.

Deployment onto a server means that the model can be trained automatically as new data becomes available and automatic predictions for new test data can be made as it becomes available. A server can also scale to meet the computational and storage demands of bigger data or more complicated models. Automated retraining could present problems however where the newer data is not of as high quality as the previous data. To combat this, unit tests can be created to test the performance of the newly trained model when it is asked to predict on a held out validation dataset. If these tests are passed then the new model can be integrated into the production pipeline.

Dashboards and Reports

The output of this deployed model could be fed automatically into a dashboard. The image below is a snapshot of the turnover dashboard hosted on Tableau server (taken from https://public.tableau.com/profile/starschema#!/vizhome/Fluctuationreport/Fluctuationreport, accessed 28/03/2019).



Such a dashboard could be augmented with the predictions outputted by the model I deployed into production. These predictions would give insight into those at risk of turnover so that mitigating actions can be taken by leadership and/or HR experts.

Automated reports could also be sent to managers and/or HR experts providing an overview of which employees are at risk of turnover, along with recommendations on retention strategies tailored to each employee.

Employees could also be classified into how much of a potential loss, both tangible and intangible, their departure will have on the business as a whole. For example, a low level employee, with little responsibility and business knowledge, leaving the company would have a much smaller impact than an employee in the senior leadership leaving. It is critical that steps are taken to reduce the risk of turnover, particularly when the costs of that turnover will be incredibly high to the business as a whole.

Issues to Consider

Deploying the turnover prediction model to production and integrating its outputs with dashboards and reports will require careful planning especially around information security and validation. This is sensitive information and it will need to be decided who should have access rights to this data. Careful validation of any model put into production will need to be done by both the data scientist as well as HR and business experts.

Another issue to consider is that incorporating up-to-date data in the training dataset could corrupt the model, i.e. up-to-date data will include employees who are just about to leave the company but they will be flagged as no turnover. The model will see these records as no turnover when in fact they should be recorded as the opposite. It is therefore important that data with a slight lag is used to train the model but with an up-to-date “has_left” feature.

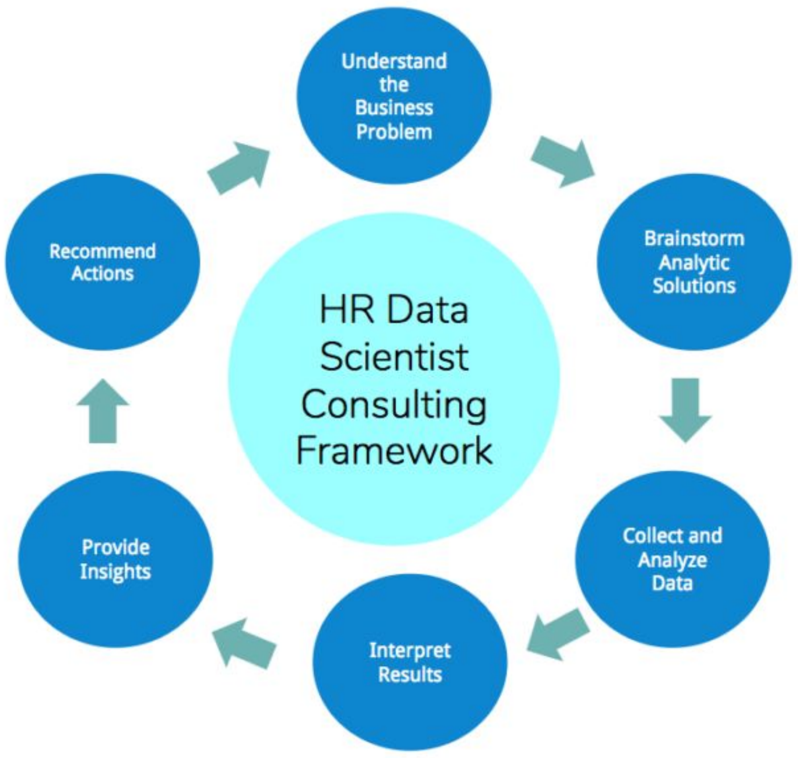
Finally, when a model is in production and steps are being taken based on its predictions, all future models will be trained on data the old model has influenced. Any future model will have to be assessed in the knowledge that the older model has influenced its training data. To get uninfluenced training data, we would have to ignore a subset of the predicted turnover cases. This could obviously have a huge negative cost to the company, so awareness of this issue needs to be clearly made.

Further Work

Before deploying to production I would want to experiment further with different models, hyperparameter and data preparation combinations in an effort to increase performance. I would also like to include more features in the input dataset (several of these features may already be included but due to the anonymisation and obfuscation of feature names I cannot be certain):

- Overtime, time since last promotion, time at the company, distance from home, work life balance, business travel, number of children, department/team name, age, education level, field of education, years of experience, job involvement, job satisfaction, performance rating, monthly income, when new practices to reduce turnover were introduced, etc.

Lastly, one thing missing from this assignment was knowledge of the business and its HR practises and policies. This kind of business understanding is essential for machine learning models to be effective. With strong business understanding, new data can be collected and new features can be engineered which will massively improve model performance.


HR Data Scientist Consulting Framework
Understand the Business Problem
Brainstorm Analytic Solutions
Collect and Analyze Data
Interpret Results
Provide Insights
Recommend Actions

Appendix

Preprocessing

Using the knowledge gained from the exploratory data analysis, I used several preprocessing steps in order to prepare the data for modelling.

Resampling

One approach I used to address the class imbalance of the target variable, “has_left”, was to use resampling techniques on the data to bring the class balance between Turnover and Not Turnover back to equality, i.e. have the same number of Turnover records as Not Turnover records.

I used random oversampling of the minority class to create a balanced training dataset. Random oversampling is a technique in which observations belonging to the minority class are duplicated at random until there is a balance between the number of observations in all classes. This technique is applied to training data before it is used to fit the model.

Potential downsides to random oversampling include that the model will overfit the training data. Specifically, because the records which were oversampled do not add any new information, they merely repeat information which already exists in the data, it causes the model to place a greater importance on oversampled records when in reality they may not generalise well to unseen data.

Other approaches for resampling which were not explored due to time constraints include undersampling and SMOTE (Synthetic Minority Over-sampling Technique). Undersampling is similar to oversampling, except records in the majority class are removed until there is balance between the records in all classes. SMOTE is a technique whereby new records in the minority class are created artificially until the number of records in all classes is equal.

It is also possible to deal with the class imbalance algorithmically by using, for example, the class weight hyperparameter in Random Forest Classifiers.

Feature engineering

Feature engineering is the process of creating new features from existing data. Good feature engineering is commonly how massive gains in the predictive power of models are made.

I experimented with several feature engineering techniques, outlined below, in an effort to increase the performance of the models I tested.

One-Hot Encoding

One-hot encoding is a technique where a new binary feature is created for each unique categorical value in a column.

Label Encoding

Label encoding involves assigning each unique categorical value in a feature to an number and then replacing those categorical values with their uniquely assigned number.

Principal Component Analysis (PCA)

Principal component analysis (PCA) is a technique used to reduce the dimensionality (number of features) in the data to a smaller number of features that still contain most of the information. Although some information loss does occur, a smaller number of features can lead to the machine learning model achieving greater accuracy and generalising better to unseen data.

Polynomial Features

Creating polynomial features is the process of multiplying each numeric feature by both itself and every other numeric feature in every possible combination up to a specified degree.

Subtracting Features

Subtracting features is as simple as subtracting one numeric feature from another. This linear transformation while simple, often results in large performance gains in nonlinear models such as XGBoost (discussed below).

Scaling

Linear models in particular are prone to overfitting features which are out of proportion to the other features in the training data. In order to combat this, the training data can be scaled so that all of the features conform to the same ranges of values. The scaling method I used was Sklearn’s MinMaxScaler which scales the data so that every feature contains the same minimum and maximum value.

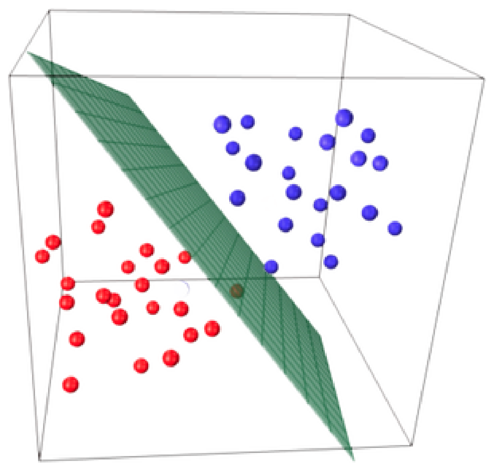
This scaler was applied to all independent features so that the data could be well used by linear models.

Models

I experimented with 3 machine learning models: 1 linear model and two nonlinear tree-ensemble models.

Logistic Regression

Logistic regression is a linear model which outputs the probability that a value belongs to a class. The algorithm assumes that the input data can be split by a linear boundary between the classes which is either a line or a plane depending on whether the input data contains two or more features. An example of such a linear plane between linearly separable classes is shown below.



Unlike the nonlinear models discussed below, Logistic Regression is more interpretable and allows for inferences to be made. For example, with an accurate Logistic Regression model, we could examine its coefficients to determine how much an increase in salary affects the probability that an employee will turnover.

The main disadvantage of using a linear model like this is that if the relationships between the data are nonlinear, it will struggle to perform to the standards of the more state-of-the-art nonlinear models such as the two discussed below.

Random Forest

Decision Trees are a machine learning approach used to discreetly categorize data by forming a structure of branches and leaves, where the branches represent logical junctions based on predictors and the leaves represent class labels. Random Forests build on this by generating numerous Decision Trees and then outputting the mode output of those individual trees. Random Forest is among the best performing classifiers and is particularly strong at minimising overfitting.

Extreme Gradient Boosting (XGBoost)

XGBoost, like Random Forest, is a tree ensemble model which outputs the most frequent result of multiple weak decision tree prediction models. Boosting algorithms add new trees sequentially, without altering existing trees, to correct the errors made by the existing trees until no further improvement can be made or a fixed number of trees has been reached.

Gradient Boosting uses the gradient descent algorithm to optimise parameters in
order to minimize the residual loss function when adding new trees. Thus, gradient descent is not used to optimize parameters, but instead is used to optimize the residual loss when a new tree is added to the model, i.e. the new tree is parameterised and the parameters of the tree are altered so that the residual loss is minimized. Thus, a cost function is optimised by gradient descent in function space through an iterative process which chooses a function that points in the negative gradient direction.

Model Evaluation

Train Test Split

The evaluation of machine learning classification models generally involves measuring the models performance when it is used to make predictions on an unseen set of data. In this case, I trained the model with 2/3rds of the data and kept back the remaining 1/3rd of data solely for the purpose of evaluating the models performance.

Cross Validation

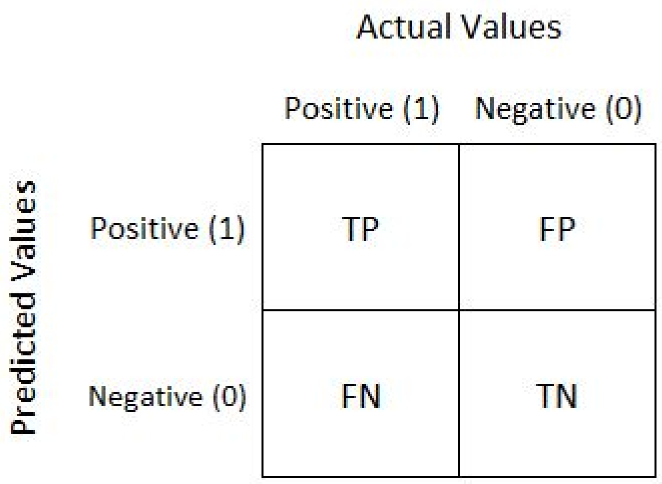
Another strategy for splitting data into training and testing sets is cross validation. Unlike with a train test split approach, cross validation splits the input sample into equal size subsamples. One of the subsamples is retained for testing while the other subsamples are used to train the model. This process is then repeated until every subsample has been used for testing.

I used the Sklearn implementation of random search with cross validation to tune the model hyperparameters for the recommended XGBoost solution. In the future and with more time, cross validation is a method which should be considered when evaluating the models.

Confusion Matrix

The confusion matrix is made up of the True Positive, True Negative, False Positive and False Negative results outputted by a predictive classification model.

A True Positive is when the model correctly predicted a positive record and a True Negative is when a correct prediction is made on a negative record. A False Positive is when the model incorrectly predicted a record as positive and lastly a False Negative is when the model incorrectly predicted a record as negative.



Metrics

I considered the following metrics when evaluating the models:

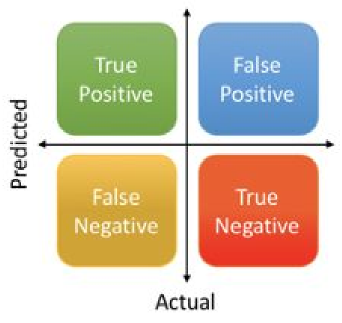
1. Accuracy
2. Precision
3. Recall
4. Average Precision
5. Area Under the Receiver Operator Curve

Accuracy, Precision and Recall

$$\textbf{Precision} = \frac{\text{True Positive}}{\text{Actual Results}} \text{ or } \frac{\text{True Positive}}{\text{True Positive + False Positive}}$$

$$\textbf{Recall} = \frac{\text{True Positive}}{\text{Predicted Results}} \text{ or } \frac{\text{True Positive}}{\text{True Positive + False Negative}}$$

$$\textbf{Accuracy} = \frac{\text{True Positive + True Negative}}{\text{Total}}$$



Average Precision

Average Precision summarizes a precision-recall curve as the weighted mean of precisions achieved at each threshold, with the increase in recall from the previous threshold used as the weight. The Sklearn implementation is not interpolated and is different from computing the area under the precision-recall curve with the trapezoidal rule, which uses linear interpolation and can be too optimistic.

Feature Importance

After a linear or tree-based model is trained, it is possible to extract from it how important each feature was for influencing its predictions. These importances sum to 1, with a larger value indicating that the variable was more important to the model’s predictions.