Modelli

Preprocessing - data subsetting

Given that models can work with different type of datasets, different preprocessing pipelines has been developed.

After discussion with the team, we've decided for using a subset of the whole dataset. In particular, the following choices have been made:

Feature selection and transformation

Regional subsetting

EDA revealed that HGF firm distribution across country is irregular, with a range from ~2% to ~35% of total firms. This is probably due to some sampling bias that we cannot control. Since our aim is to obtain a predictive model for italian startups, only data from Italy has been used.

Composed variables

in Balance Sheets and Income Statements, many covariates are given by the linear combination (usually sum or difference) of other variables (usually 2 or 3). For this reason all composed covariates has been dropped.

Redundant and non-contextualized columns

For some analysis, all geolocalized data has been dropped, for this first part of the modelling.

Small, private companies only

Public companies data were not included in the analysis. Additionally, it turned out that some "newborn" companies were actually name changes of mature companies. With the aim of being able to infer our models on SMEs/Startups, we selected only Small Companies.

US Census Bureaus, in 2012, published data stating that 98.2% of companies, at the beginning, has less than 20 employees. This number drops to 94% after 1 year, 92.4% after 2 years. In our dataset some companies had 500+ employees. We filtered out all companies with 20+ employees.

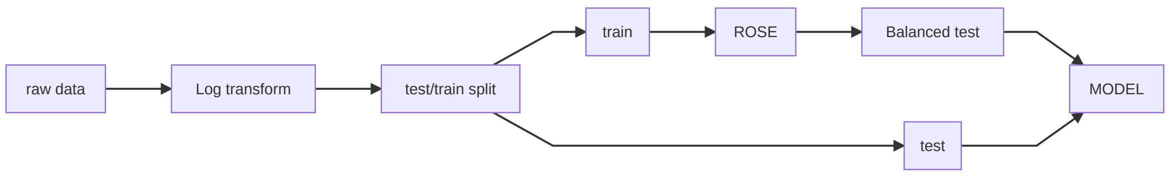
Categorical data

Most categorical data has been dropped for this first part of the modelling. It will be included later.

Incomplete data

NA's has been dropped in the first part of the analysis. Missing values imputation (with ROSE) has been performed.

ROSE (Random OverSampling Examples)

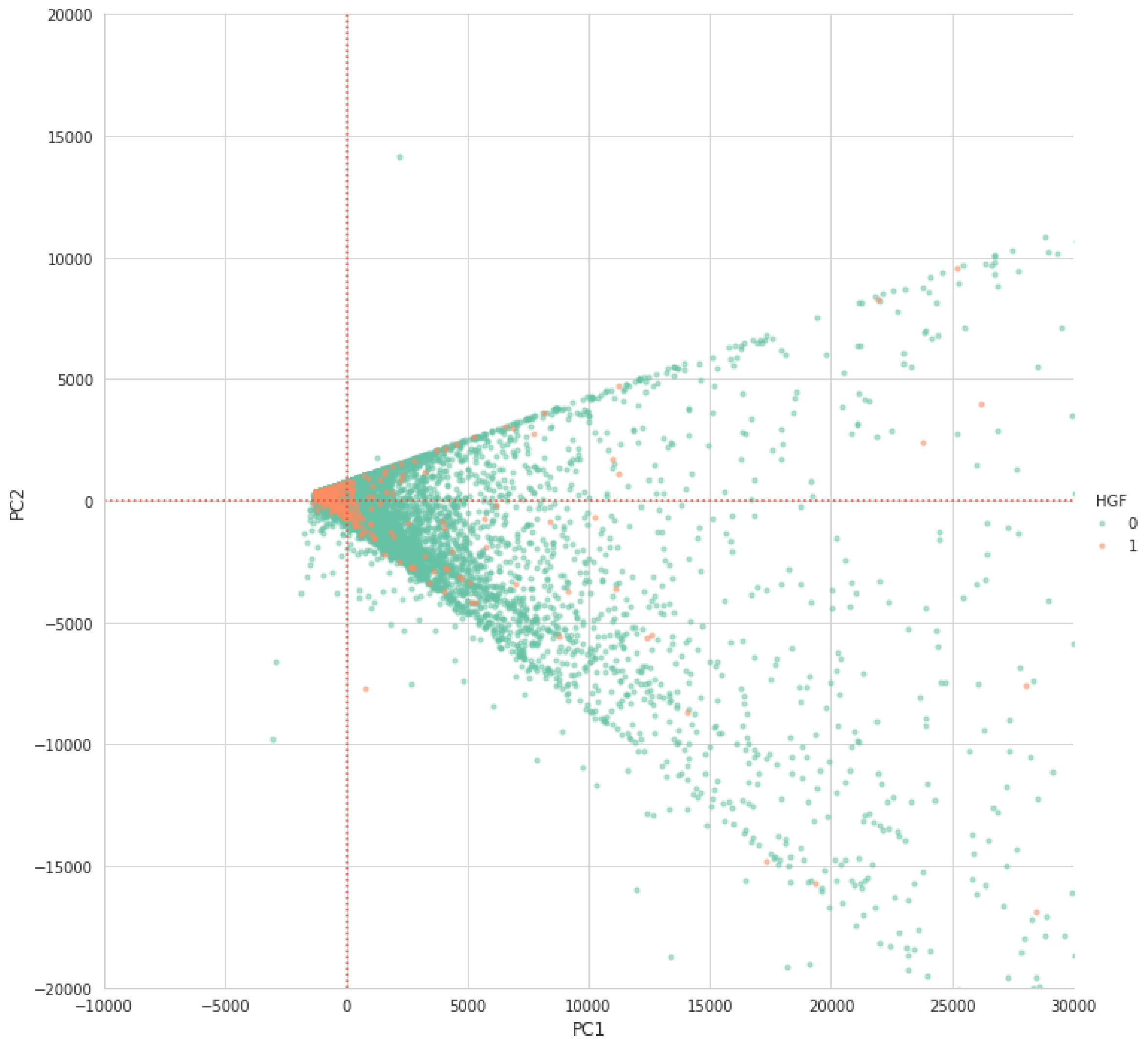


Models

Clustering and embedding

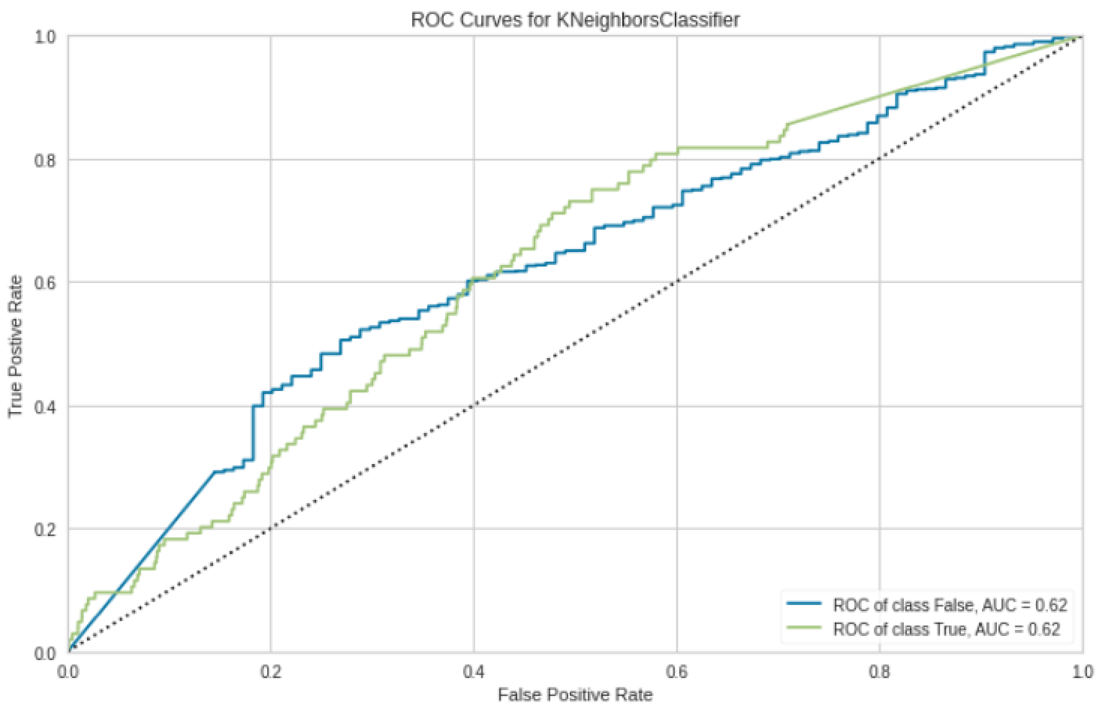
PCA

At first I tried a 4-components PCA embedding, to see if there is plain separation between groups. A negative first component could be a good indication of high $p(HGF)$.

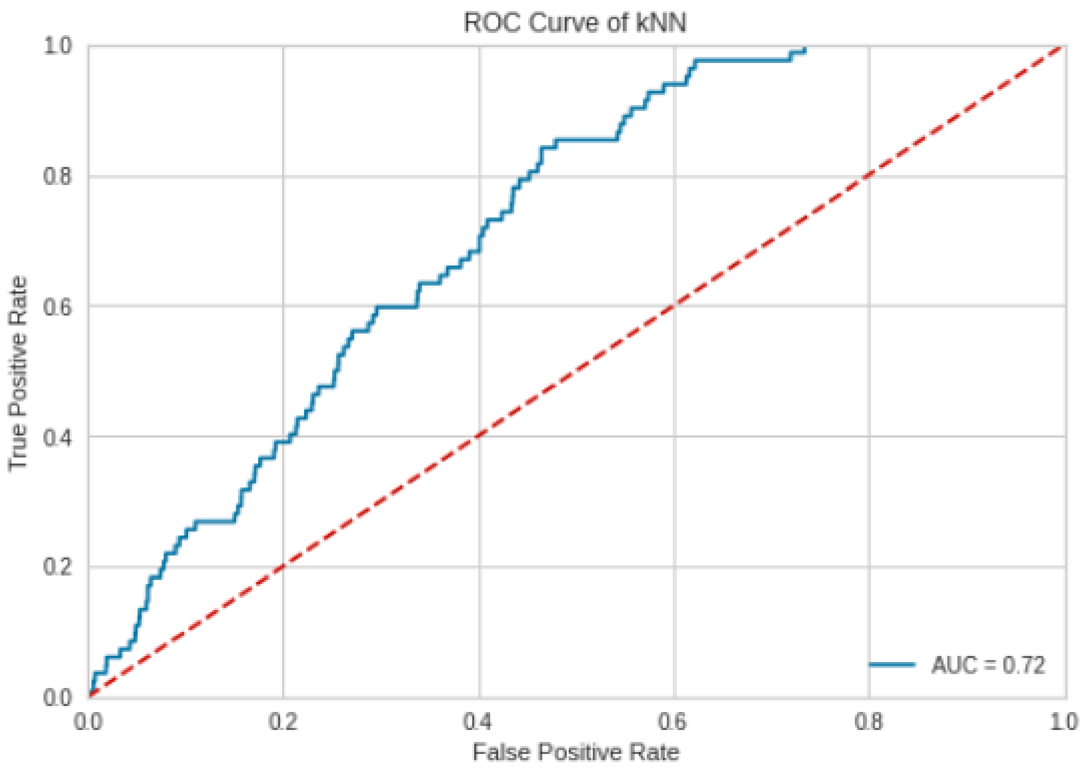


K-Nearest Neighbors classifier

- Preprocessing: `sklearn.preprocessing.StandardScaler()`
- Algorithm: `KNeighborsClassifier(n_neighbors=50, p=2, weights="distance")`
- Results:



After balancing our model with ROSE, the performance increased to

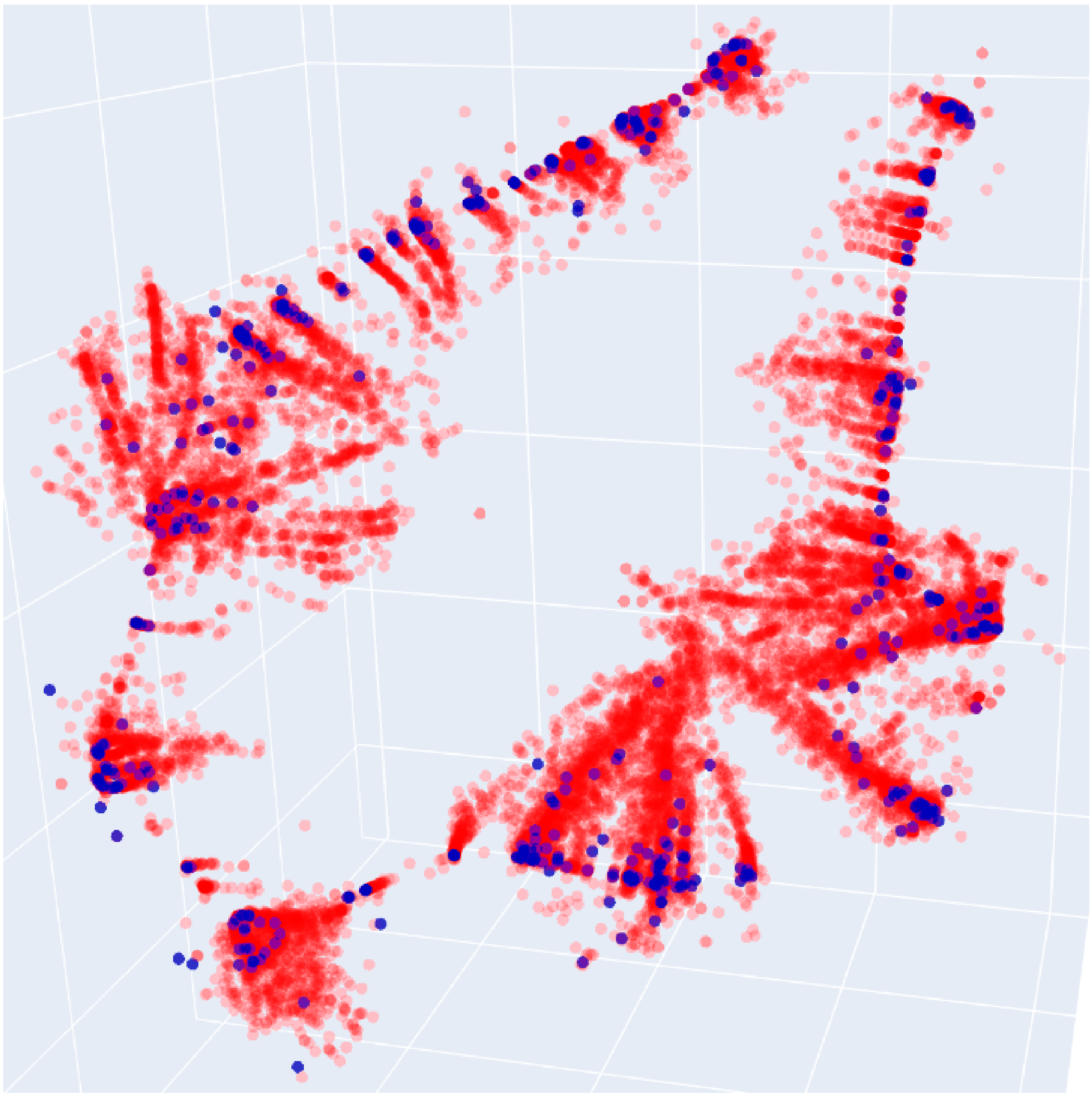


t-distributed Stochastic Neighbor Embedding

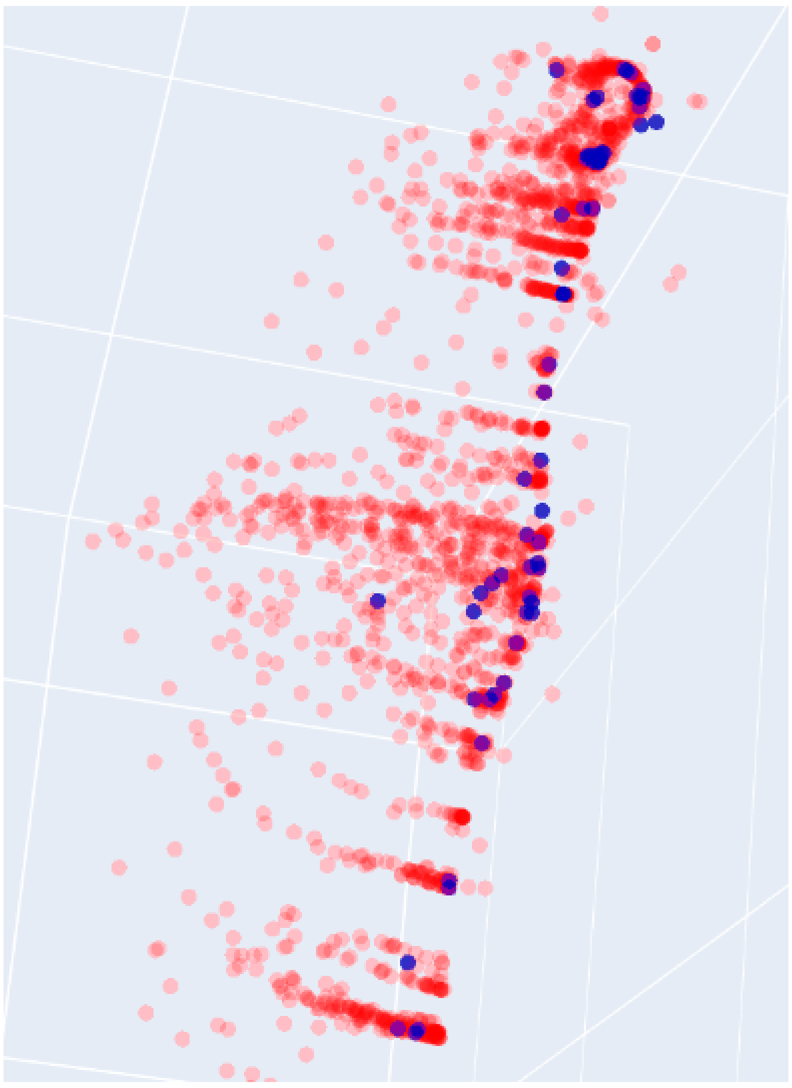
t-SNE converts similarities between data points to joint probabilities and tries to minimize the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data.

Projecting data points on a 3 dimensional manifold led to an interesting observation. See attached `tsne.webm` for a video representation of the embedding.

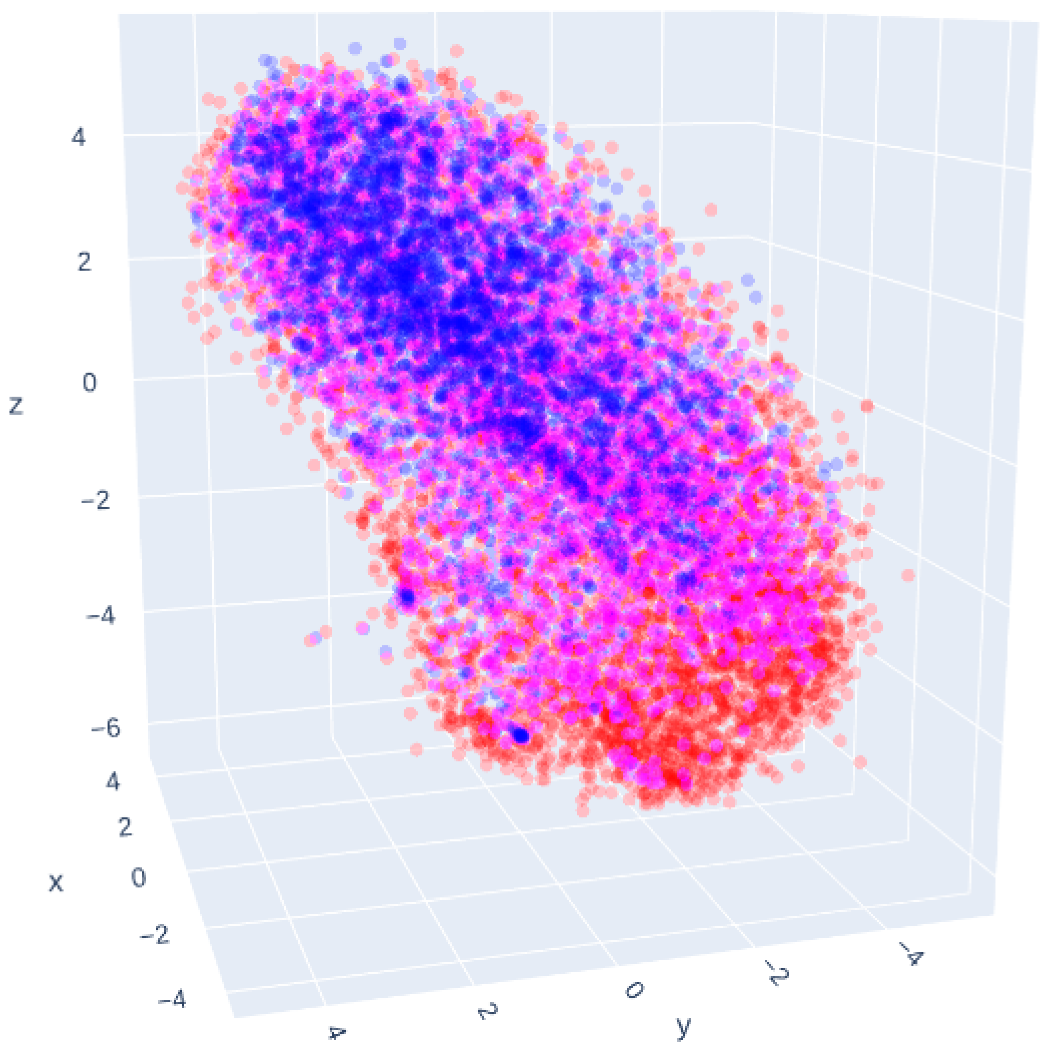
Data seem to cluster in strings along paths, aligned along a "U" shape. Along these strings, HGF firms appears almost exclusively on the same end. We have to remember that t-SNE manifold is not euclidean, and understand if we could use this property to classify.



In this figure, we see a 2D projection of the embedding.



A particular of the strings. Blue points are HGF firms.
When augmenting data with ROSE, this pattern disappears, and the points appear to be distributed in a multivariate normal. Still, in the cloud, one can observe a gradient of frequency of HGF firms.



Classification

Linear/Quadratic Discriminant Analysis , Gaussian Naive Bayes Classifier

- Preprocessing: only numeric columns has been considered in the analysis.
- Interaction features: Polynomial interaction terms of degree 1 and 2 has been added to the dataset, up to a total of 4494 columns.
- Before fitting, `StandardScaler()` was used to normalize data. Variance Inflation Factor analysis (from `statsmodels.stats.outliers_influence.variance_inflation_factor`) has been used to deal with multicollinear features.
- Hyperparameters optimization: grid search with 5-fold cross validation.
- Results (accuracy):

	LDA	QDA	GNBC
Best result	0.487	0.509	0.512

Discussion: class imbalance effect doesn't forgive, in those models.

A less naive Bayes Classifier

Inspired by ROSE, I tested the code for a KDEClassifier. This is not included in common libraries. Here is the code:

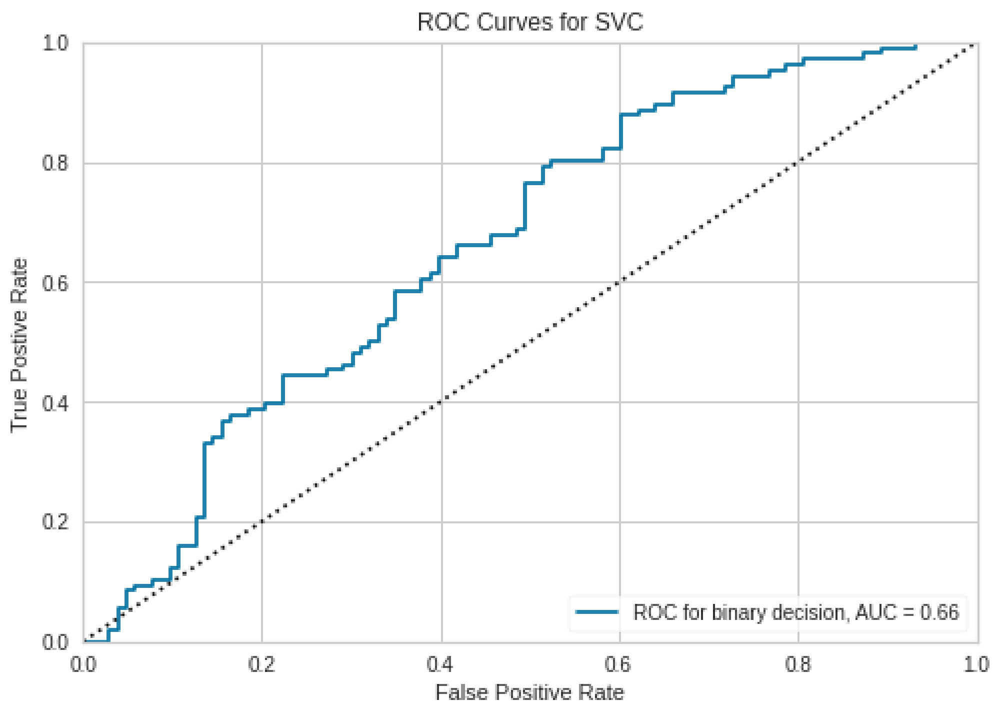
```python
from sklearn.base import BaseEstimator, ClassifierMixin


class KDEClassifier(BaseEstimator, ClassifierMixin):
    """Bayesian generative classification based on KDE

    Parameters
    ----------
    bandwidth : float
        the kernel bandwidth within each class
    kernel : str
        the kernel name, passed to KernelDensity
    """
    def __init__(self, bandwidth=1.0, kernel='gaussian'):
        self.bandwidth = bandwidth
        self.kernel = kernel

    def fit(self, X, y):
        self.classes_ = np.sort(np.unique(y))
        training_sets = [X[y == yi] for yi in self.classes_]
        self.models_ = [KernelDensity(bandwidth=self.bandwidth,
                                      kernel=self.kernel).fit(Xi)
                        for Xi in training_sets]
        self.logpriors_ = [np.log(Xi.shape[0] / X.shape[0])
                           for Xi in training_sets]
        return self

    def predict_proba(self, X):
        logprobs = np.array([model.score_samples(X)
                             for model in self.models_]).T
        result = np.exp(logprobs + self.logpriors_)
        return result / result.sum(1, keepdims=True)

    def predict(self, X):
        return self.classes_[np.argmax(self.predict_proba(X), 1)]
```

Bandwidth parameter has been set with a cross validated grid search, but ROC-AUC values are insignificant (~0.5).

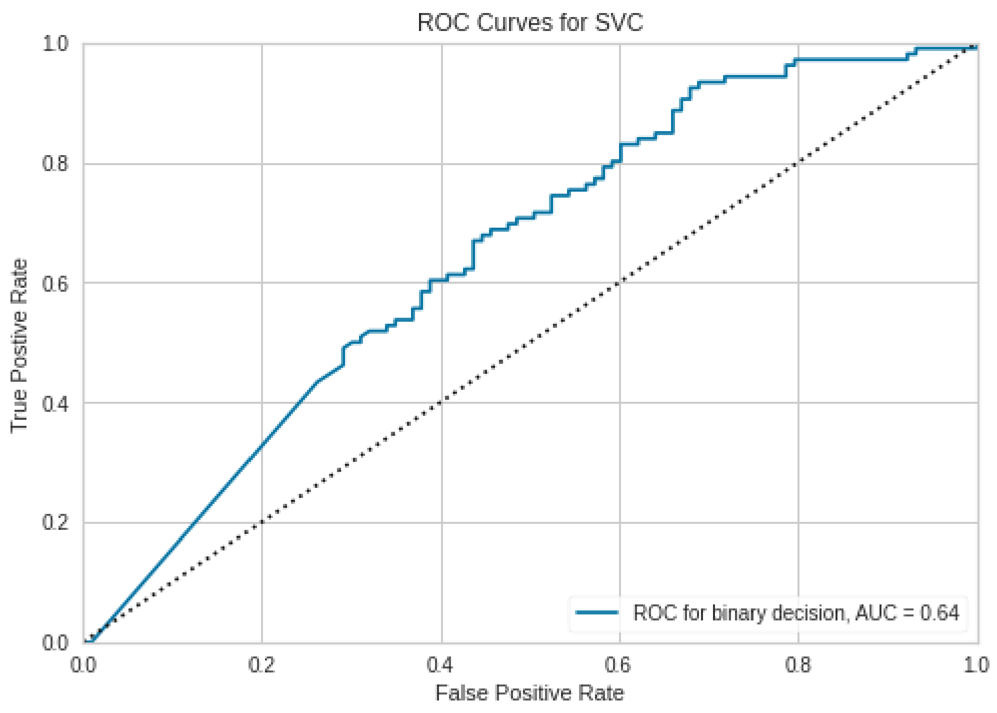
Support Vector Classifier

Different kernels has been tested, with the following performances:

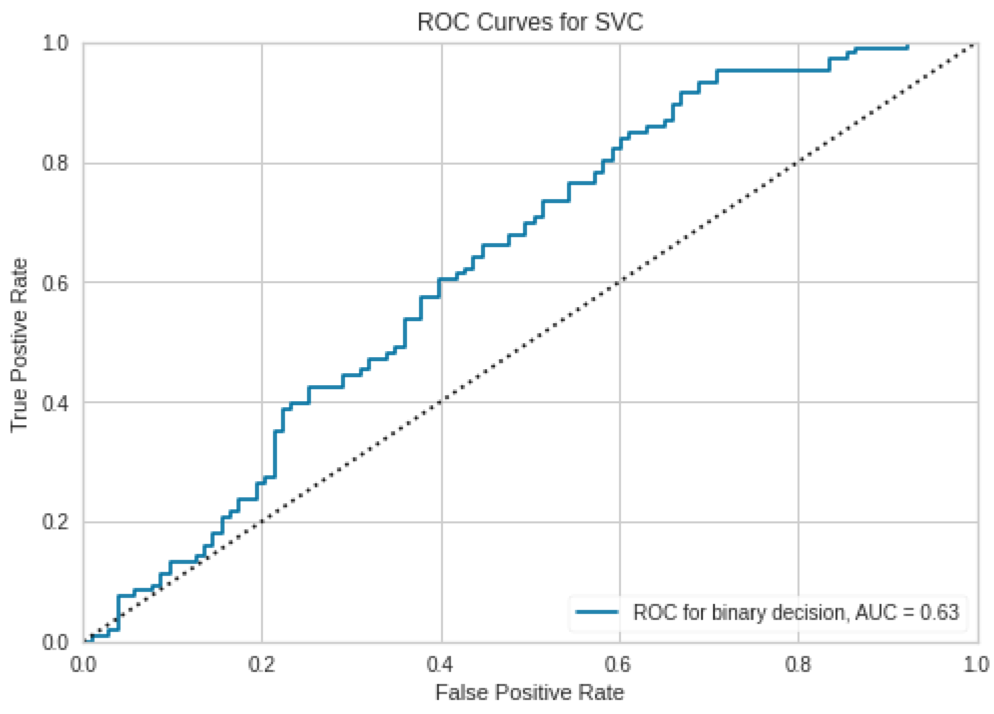
Linear Kernel



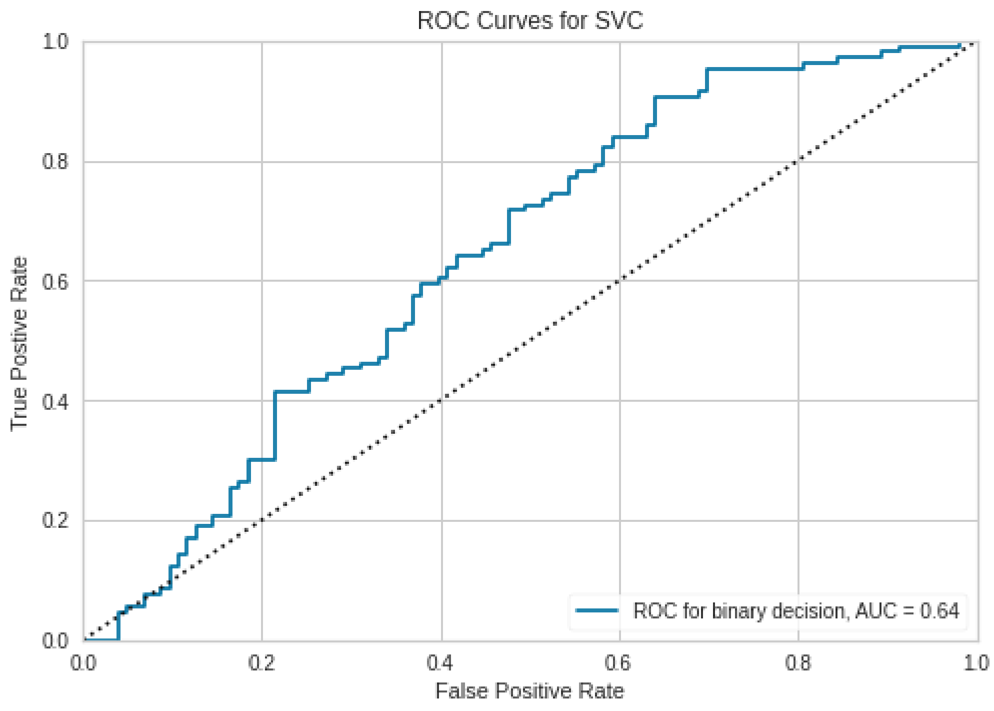
Poly(8) Kernel



Gaussian Kernel



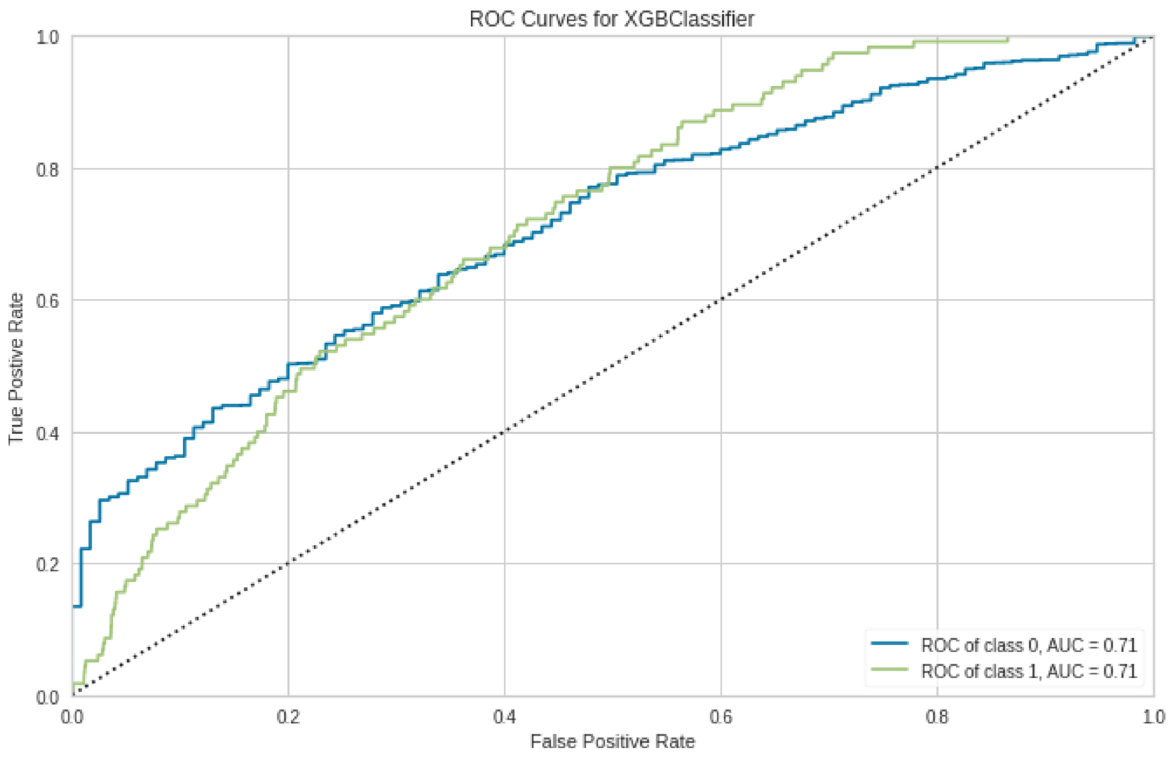
Sigmoid Kernel



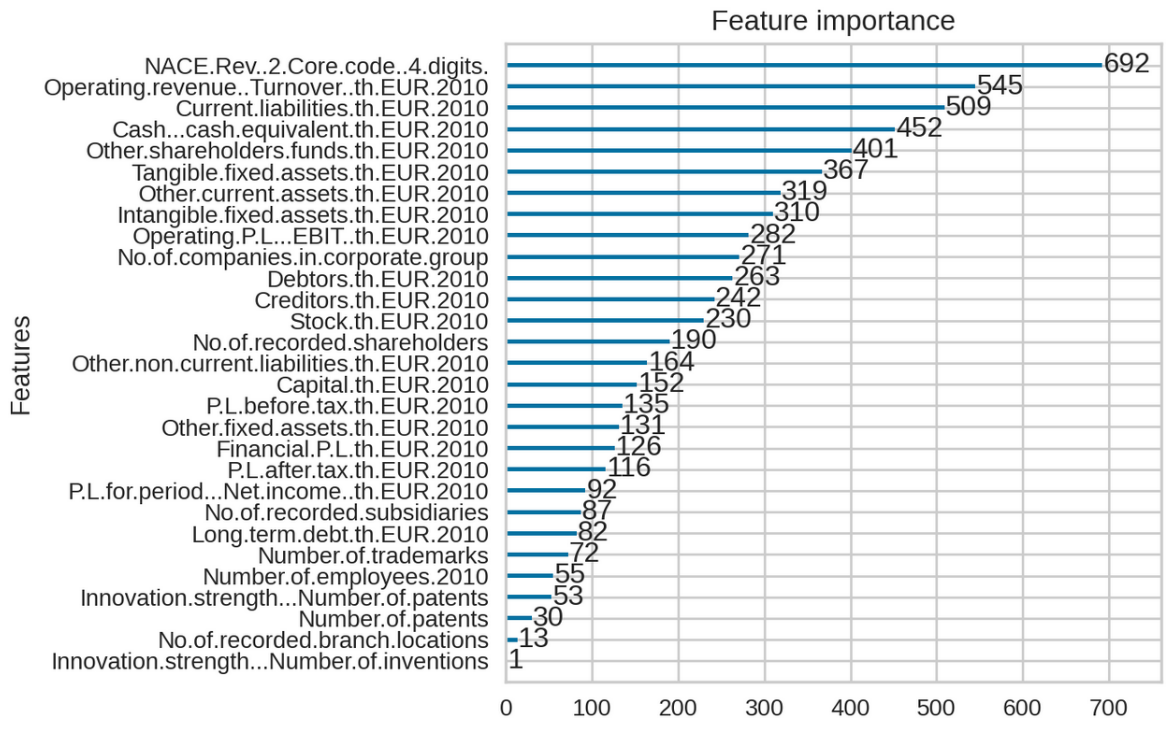
SVM training with nonlinear kernels is, complexity-wise, approximately $O(N^2_{samples} * M_{features})$, so it has proven very slow for large datasets.

XGBoost

As usual, grid search + CV was used to optimize hyperparameters. The following figure shows the performance of my best model, out of ~5000.



Feature importance in XGB:



Neural Network

A fully connected NN has been tested. Then, ROSE has been used to balance the dataset. A simple network has been trained. Dropout layers has been added between hidden layers, for regularization. Binary crossentropy was used as loss function.

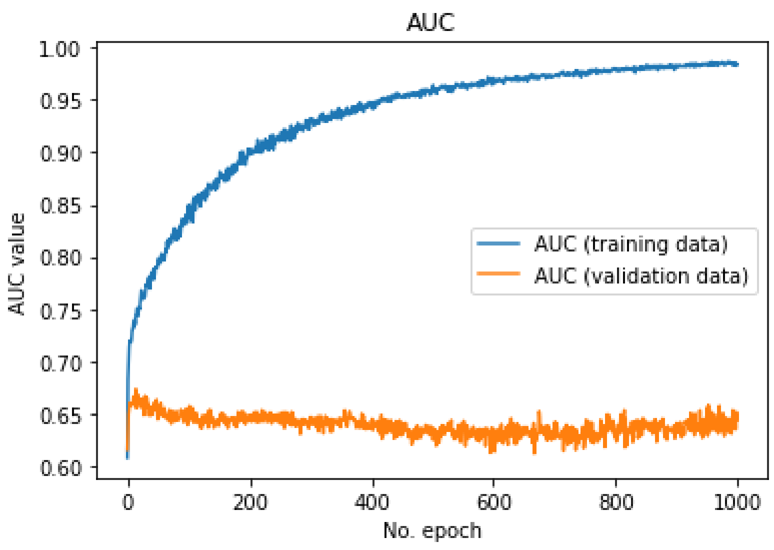
$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i * log(p(y_i)) + (1-y_i) * log(1-p(y_i))$$

Network structure:

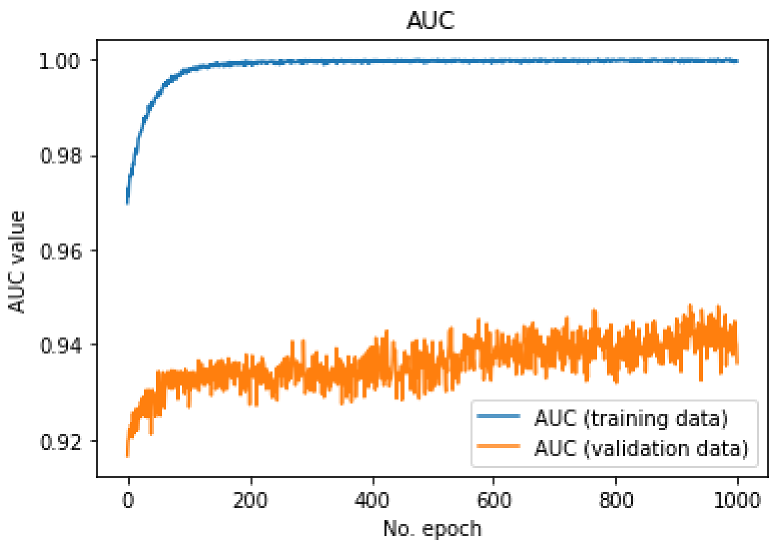
```
model = Sequential()
model.add(Dense(1000,  activation='relu',kernel_initializer="glorot_uniform",))
model.add(Dropout(0.2))
model.add(Dense(2000, activation='relu', kernel_initializer="glorot_uniform",))
model.add(Dropout(0.2))
model.add(Dense(1000, activation='relu', kernel_initializer="glorot_uniform",))
model.add(Dense(1, activation='sigmoid'))

opt = keras.optimizers.Adam(learning_rate=1e-5)

model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['AUC'])
```

Pro tip: ROSE generated data are ordered by label. First all HGF=0, then all HGF=1. Keras, when using internal cross-validation, doesn't split the data, so only a single label will be used. This was long to debug. A pre-train_test_split shuffle was sufficient to fix.

Results: the same network was trained with the unbalanced dataset and ROSE balanced data. With unbalanced data, validation set ROC-AUC stabilizes around ~0.65. ROSE improves the situation, leading to a validation ROC-AUC of 0.94.

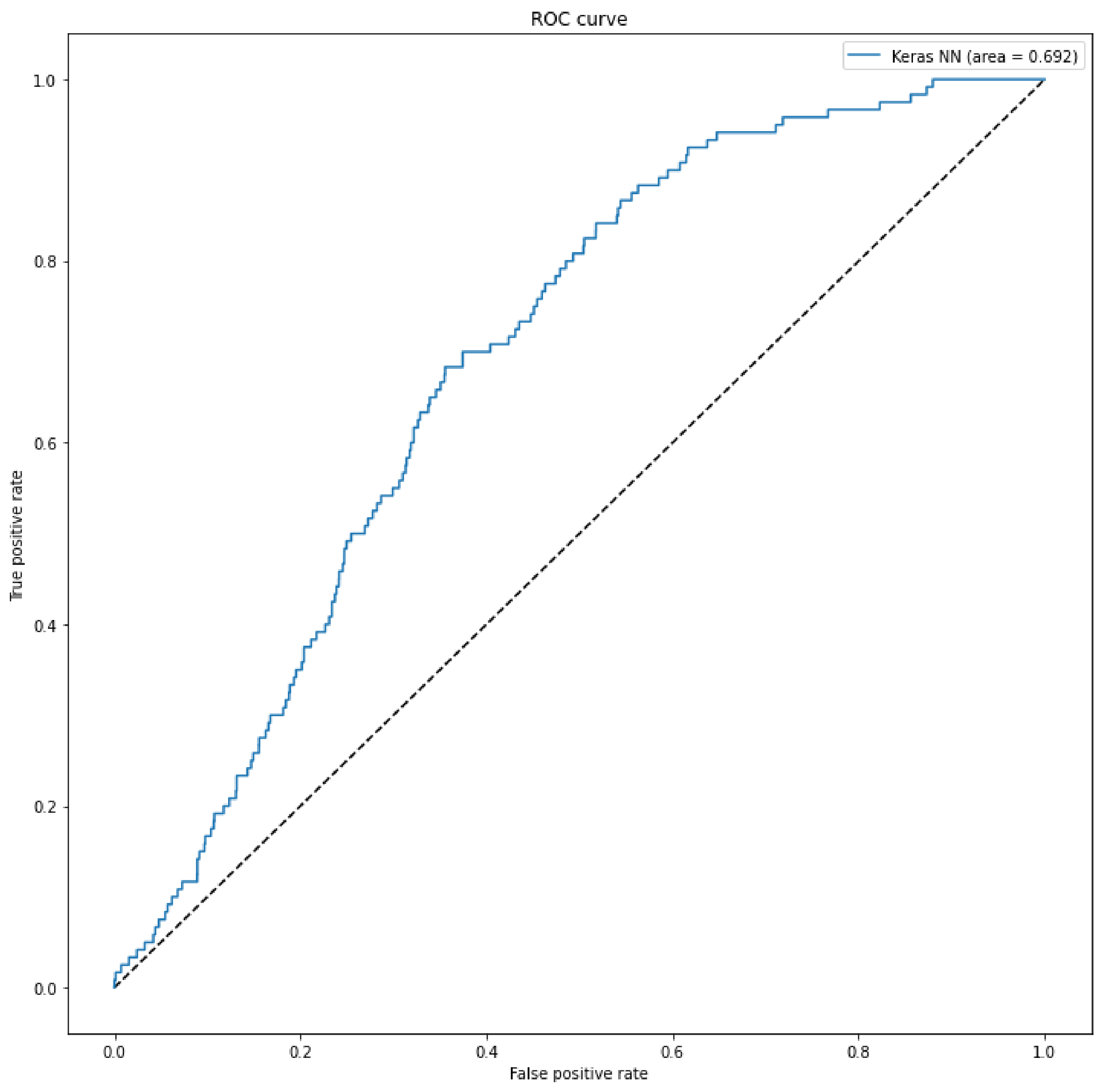


Non-balanced dataset training diagram.

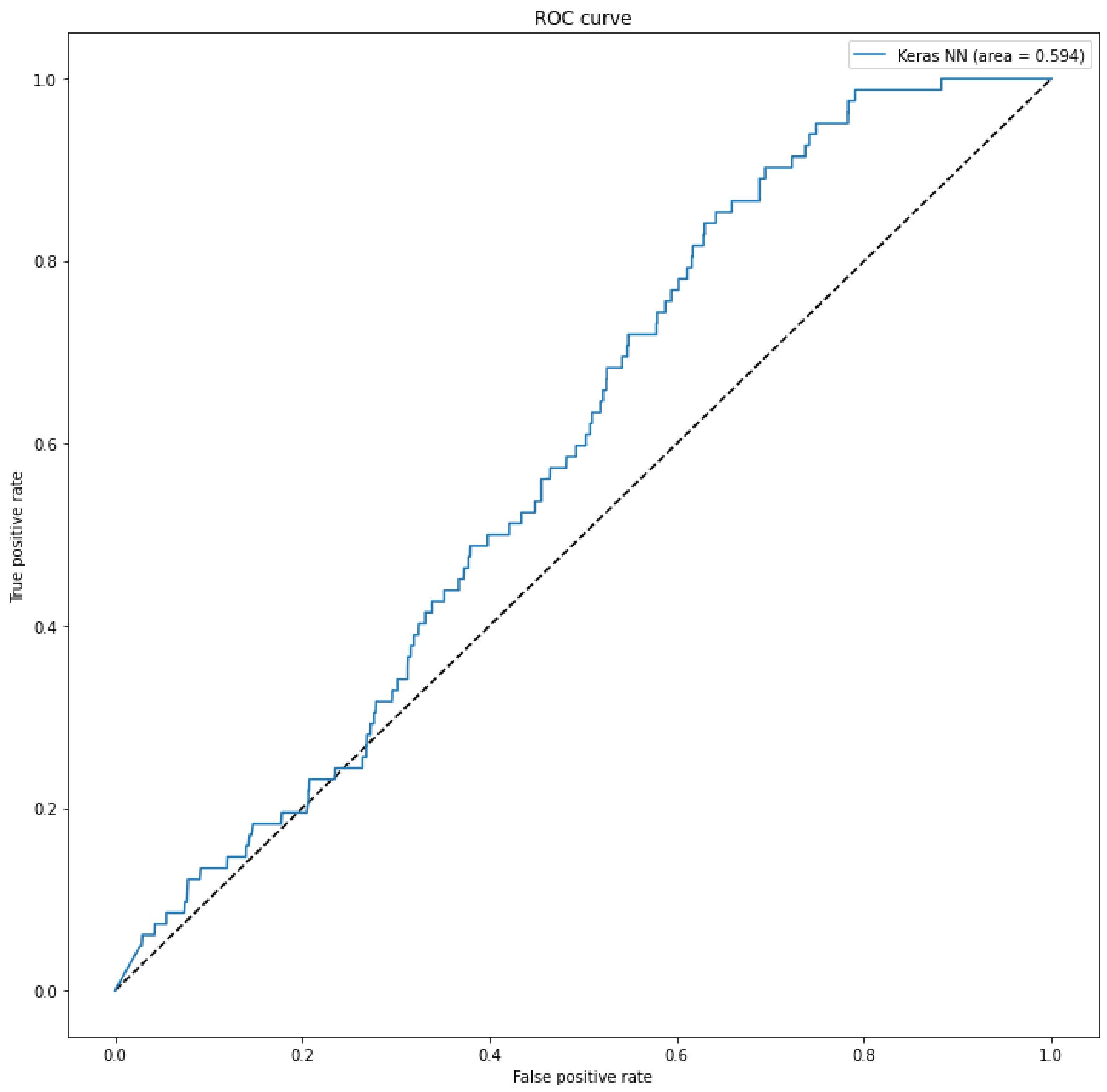


ROSE Dataset NN training diagram. Note the different y_scale.

Still, when tested on pre-ROSE data, inference is bad.



ROC curve for unbalanced dataset



ROC curve for ROSE balanced dataset.

Genetic programming

TPOT

I used genetic programming platform TPOT (Tree-based Pipeline Optimization Tool, link). Command line tool was used, to take advantage of parallelization.

```
tpot ../logs.csv # work on transformed data log(data-min(data)+1)
   -is , # separator
   -target "HGF"  # label column
   -o "tpot_exported_pipeline-2.py" # exported python pipeline
   -g 20 # number of generations
   -p 50 # population size
   -scoring "roc_auc" #score metric
   -cv 5 # cross validation
   -njobs 12 #cores to be used
   -maxtime 60 #max time to perform analysis
   -s 666 # random seed
   -template 'Selector-Transformer-Classifier' # pattern for a population member
   -v 2 # verbosity
   -mode "classification" # target task
```

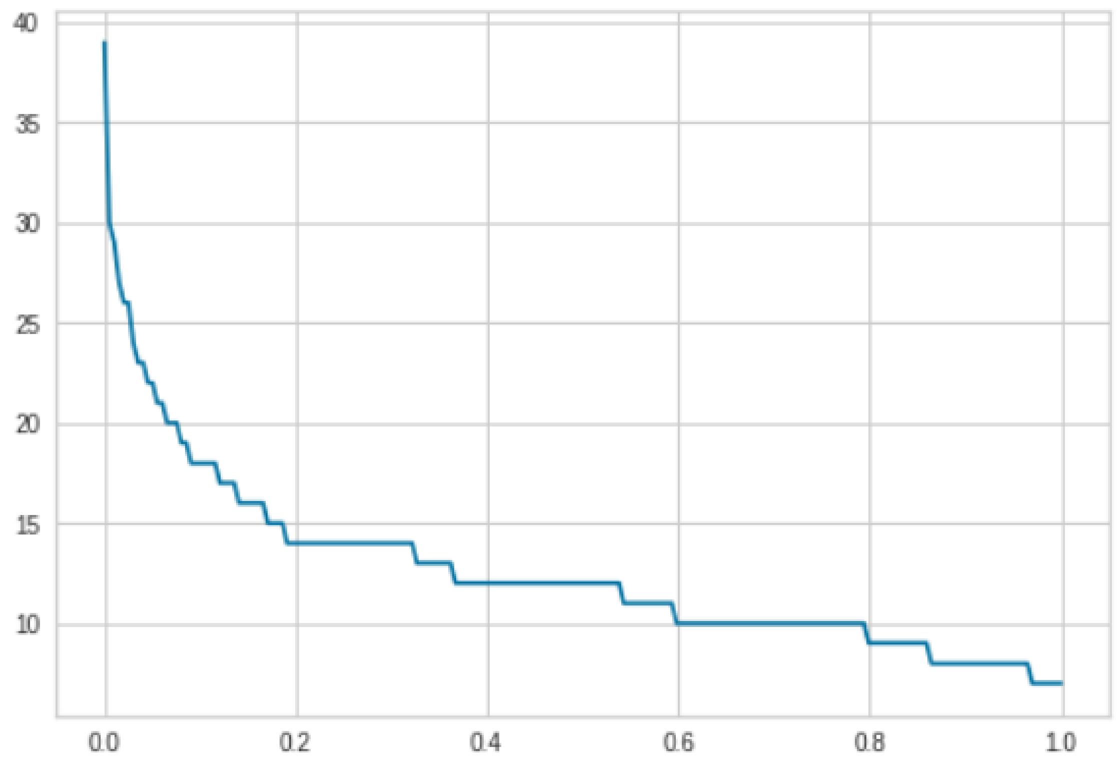
In many runs, different solution evolved:

```
Best pipeline: XGBClassifier(OneHotEncoder(VarianceThreshold(input_matrix,
threshold=0.005), minimum_fraction=0.1, sparse=False, threshold=10),
learning_rate=0.1, max_depth=6, min_child_weight=12, n_estimators=100,
nthread=1, subsample=0.4)

Best pipeline: KNeighborsClassifier(PCA(VarianceThreshold(input_matrix,
threshold=0.05), iterated_power=2, svd_solver=randomized), n_neighbors=74, p=1,
weights=uniform)

Best pipeline: ExtraTreesClassifier(RobustScaler(SelectPercentile(input_matrix,
percentile=95)), bootstrap=False, criterion=entropy, max_features=0.55,
min_samples_leaf=14, min_samples_split=13, n_estimators=100)
```

In most cases, ROC-AUC stopped growing over 0.79-0.82.

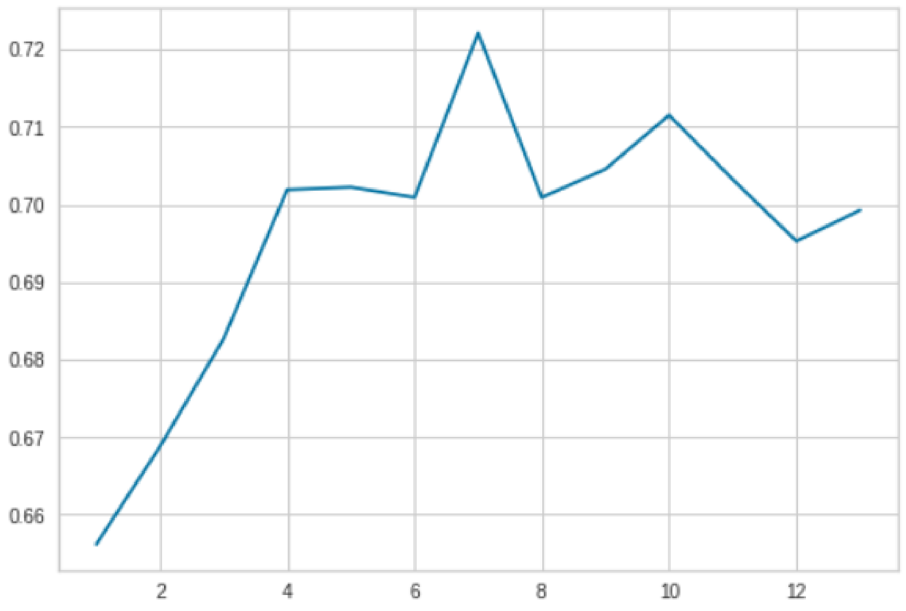
Variance threshold was often selected, so I tested a different pipeline from one of the evolved ones. At first, I selected features by their variance, plotting number of surviving features vs different variance thresholds.

```
'# of feature selected VS variance threshold'
```



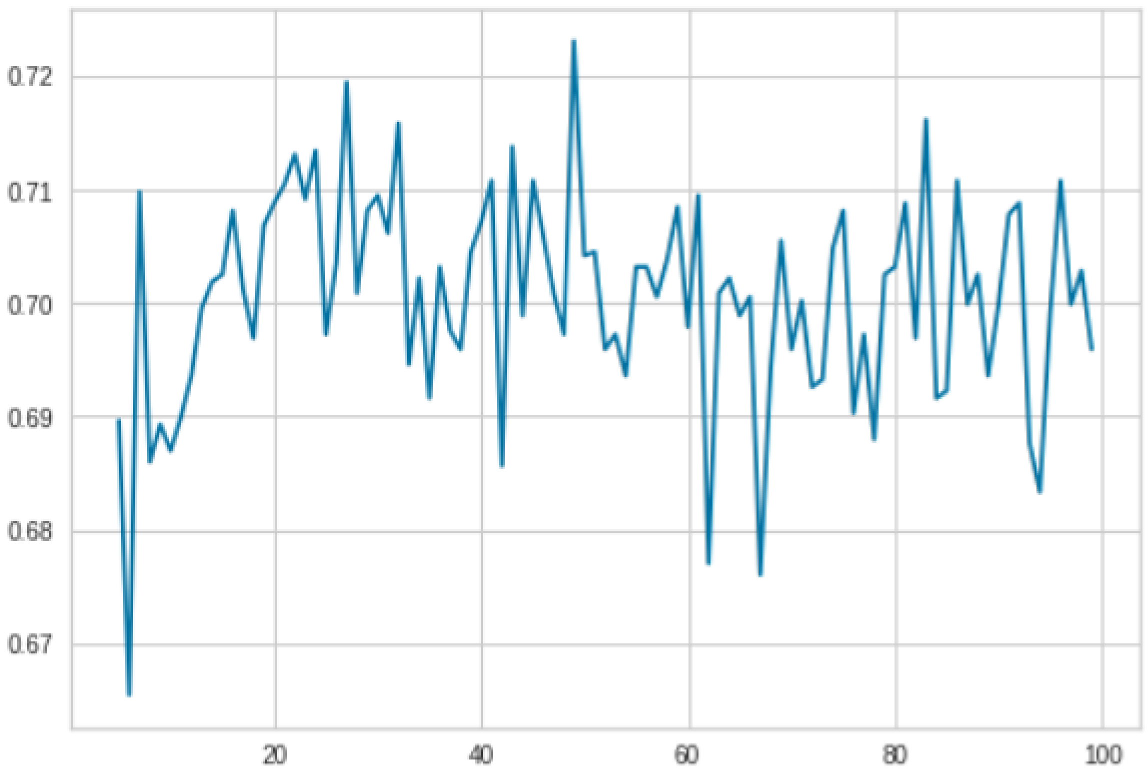
There is no evident elbow, so I selected an arbitrary threshold of 0.2. I then performed many PCA on selected features, to further reduce dimensionality. For every value of n_components, I fitted a KNN with 70 neighbors and evaluated the performance.

```
'KNClassifier score vs # of PCA components. Max = 0.7220934084133819'
```



Why 70? Fixing the number of components to 7 and varying the number of neighbors showed that there is no big influence. An example here:

```
'score VS n. of neighbors, for 7 PCA components'
```



Conclusions (so far)

- Unbalanced dataset confirms a problem when training models
- ROSE works pretty well to balance the dataset, but doesn't seem to perform well for heavily zero-inflated distributions. Changing kernel smoothing matrix multipliers didn't change the behavior in a sensible way.
- Most models' performance is capped around a ROC-AUC < ~0.7-0.8 on test set.

- Pipelines generated by evolutionary computing performs pretty well, despite being very computational intensive.