STØTTET AF
Mælkeafgiftsfonden

Report

December 16, 2022

1 DD Management: Predicting hoof disorders ('klovlidelser') in cows

This notebook details an experimentation with a machine learning model for predicting hoof disorders (klovlidelser) in cows.

Contact information: SEGES-datascience@seges.dk

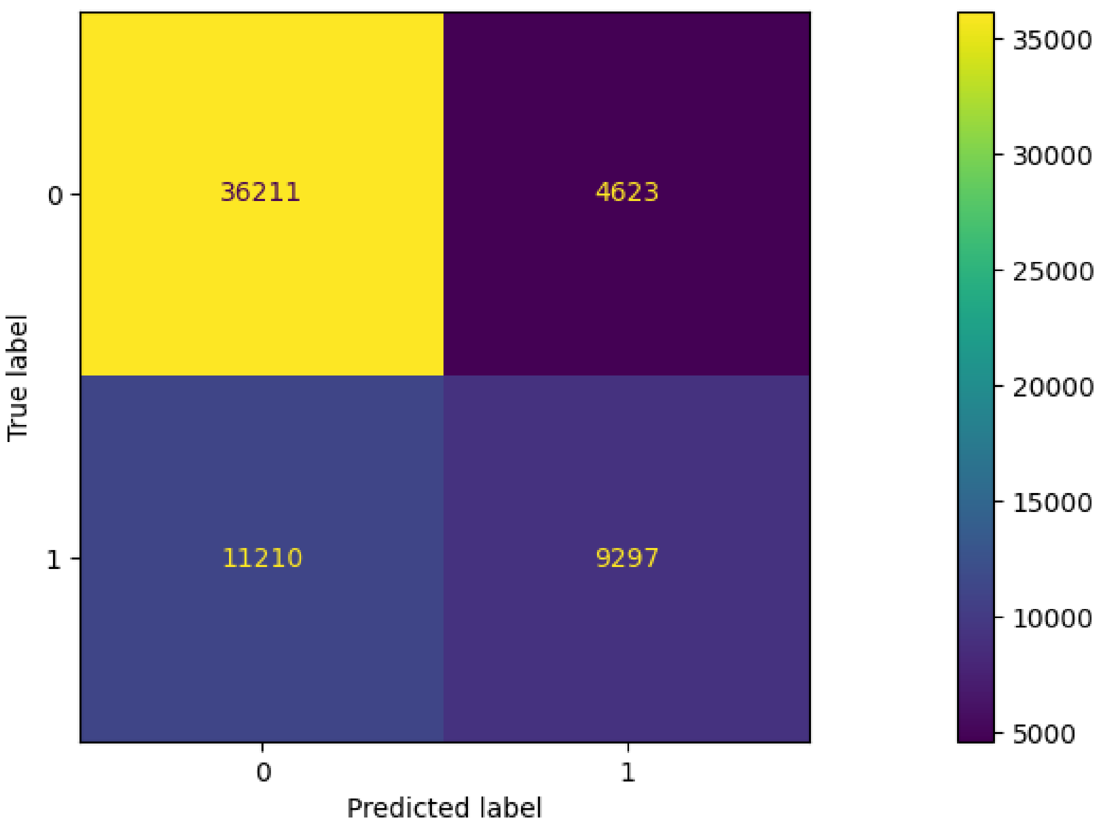
1.1 Danish Summary

Denne rapport indeholder koden der har genereret machine learning modellen til forudsigelse af klovlidelser i danske malkekøer. Target-variablen (en 0-1 variable, hvor 1 indikerer en klovlidelse) i analysen er konstrueret udfra hvorvidt en malkeko får sygdomsregistreret en klovlidelse ifbm. en klovbeskæring. Som features/forklarende variable anvendes der data fra malkerobot- og vægtmålinger, sygdomsregistreringer samt lægekontrol- og kælvningshistorik. Modellen formår på alle besætninger med et tilstrækkeligt kvalificerende datagrundlag at opnå en **ROC-AUC score på 0.78**. Under de bedste forudsætninger, på mest lignende besætninger udfra et management-perspektiv, opnås der en **ROC-AUC score på 0.83**. Variableelimering afslørerede hvilke 20 variabler der for modellen er mest forklarende ift. om en malkeko med høj sandsynlighed har pådraget sig en klovlidelse, herunder viser tidligere klovlidelser, antal kalve igennem levetid og hvor langt inde i laktationsperioden koen er sig specielt vigtige. Fine-tuning/opdatering af en delmængde af modellens hyperparametre forbedrede ikke modellens performance. Resultaterne i denne rapport indikerer at der potentielt er tilstrækkeligt signal i data til at forudsige klovlidelser i danske malkekøer. Dette giver anledning til at afprøve en POC (proof-of-concept) hvori der afsendes klovlidelsesalarmer på malkekoniveau til den delmængde af besætninger der fastholder bedst mulig eller mest lignende management praksis.

Herunder ses den forvirringsmatrix (confusion matrix) som afspejler resultatet fra eksperimentet over alle besætninger med tilstrækkeligt kvalificerende datagrundlag.

```
[ ]:
```

```
[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
     0x7fb03b6610a0>
```



1.1.1 Setup

Raw data The following data files are used across all 'besaetninger/bedrifter':

- hendelserprdag{bedrift}.csv
- dag{bedrift}.csv
- syg{bedrift}.csv
- kont{bedrift}.csv
- Kaelvninger{bedrift}.csv

Based on the weight, milk, calving, control, and disease data on the cow, the following target and features are created.

Target

- A binary target specifying whether the cow gets a hoof disorder of ['Klovbrandbyld', 'Balleforrådnelse', 'Nydannelse', 'Digital dermatitis', 'Sålesår', 'Såleblødning', 'Tånekrose', 'Dobbeltsål', 'Hul væg, byld i hvid linje'] within a week of its hoof trimming ('klovbeskæring').
- Thus, for each cow and hoof trimming, a single data point is created
- Only cows with hoof trimmings after 2019-01-01 were included for data quality purposes
- Only the cow and hoof trimming date pairs (DYR_ID, regdato) with atleast 60 days from the cow's latest calving date '(klvdato') to its hoof trimming date (klovbeskæringsdato).

- If the cow has had a horn-related hoof disorder [‘Hornrelaterede klovlidelser (Sygdomme)’] in the past 3 months, we do not include any potential hoof disorder registered at hoof trimming.
- Finally, if the farmer (‘Landmand’) has registered a ‘Klovbrandbyld’ hoof disorder at hoof trimming, it is not included as a ‘True’ target value, since farmers use this registration type to administer penicillin for any type of disorder in the animal.

Features **From dag{bedrift}.csv:**

- ydl (ydelse, Kg milk pr. day), ydlprt (protein), ydlfdt (fat), mlkgns (milking sessions pr. day).
- 10-day aggregates of the above (sum for ydelse, mean for the rest) from latest calving date (klvdato) up till hoof trimming date (klovbeskæringsdato).
- ‘ydelse60_50_days_bf_klovbskrng’, ‘ydelse50_40_days_bf_klovbskrng’, ..., ‘ydelse10_0_days_bf_klovbskrng’
- For each cow and hoof trimming date, the relative change of the above mentioned variables up until hoof trimming compared to the [60 days - 50 days before hoof trimming] level.
- ‘ydelse50_40_days_bf_klovbskrng_relative’, ..., ‘ydelse10_0_days_bf_klovbskrng_relative’
- For each cow and hoof trimming date, a standardized measure [i.e.: deviation from group mean divided by group standard deviation, (X - mean) / std.] as compared to the besaetning/bedrift is included of each of the above mentioned variables. This variable describes how much the cow varies from its besaetning on a given point in its lactation period, for a given variable.

From hendelserprdag{bedrift}.csv:

- robK_BW, 10-day aggregates (mean), including relative and standardized measures as for the ‘dag’ variables.

- ‘gold’-period length.
- Number of times the cow has had their hoofs trimmed during their latest ‘gold’-period (‘klovbeskaer_during_gold’).
- ‘bedrift’.
- race_id.
- klvnr.
- Month of the year (seasonality).

From syg{bedrift}.csv:

- The number of times the cow has had a hoof disorder previously.
- The number of times the cow has had a digital dermatitis previously.
- The number of times the cow has had a possibly correlating hoof disorder [‘Sålesår’, ‘Såleblødning’, ‘Dobbeltsål’, ‘Hul væg, byld i hvid linje’] previously.

From kont{bedrift}.csv:

- BHB, fpf, acetone, FA_DeNovo_i_TFA, FA_Mixed_i_TFA, and FA_Preformed_i_TFA from the first and last inspection of the cow in its duration from latest calving date (klvdato) to its hoof trimming (regdato), and for the first inspection after hoof trimming if the inspection is within a week of its target hoof trimming (regdato + 5 days).

From Kaelvninger{bedrift}.csv:

- Birth complications variables, including a subjective score of the birth, whether the cow had twins, had an abortion, or the calf was stillborn.
- The 'huld' score in the 'gold' period and in the month after calving.
- The time in days from when both 'huld' scores were given from the calving date.

Models The following model is used:

- Gradient tree boosting model - catboost

Performance Metrics The model is evaluated based on the following criteria:

- **Precision** = TP / (TP + FP) = number of true positives / number of true and ***false positives***
- **Recall** = TP / (TP + FN) = number of true positives / number of true positives and ***false negatives***
- **Accuracy** = (TP+TN)/Total number of cases = (TP+TN)/(TP+FP+TN+FN)
- **Precision-Recall AUC** (Area under the curve)
- **ROC AUC** (Area under the curve). ROC plots the false positive rate FPR(t) against the true positive rate TPR(t), for varying decision thresholds (or prediction cutoffs) t.

1.1.2 Experiments

1) First, a model is trained on data across the 59 'besaetninger/bedrifter' with >200 unique cows and many previous hoof disorder and treatment registrations. Model was validated on a random subset of the data (a 80/20 train/validation split of the data). Early stopping together with the validation set is used to find the optimal number of estimators (trees) in the model. The standard parameters in catboost are used.

2) Next, the same approach is used to train a model on all eligible besaetninger/bedrifter, where eligible means those that had sufficient available data from 2019-01-01 onwards.

3) The features are then examined to determine the most important features and their effect on the target. The least important features are removed and a new model is trained. This is done in 20 steps, where a small subset of features is removed for each run.

4) Hyperparameter tuning by Randomized Search Cross Validation for different hyperparameter values of the learning rate and max_depth (*learning rate*: reducing the gradient step, too high value can diverge model from global minimum, *max_depth:* depth of the tree by number of splits, the higher the value the more likely model is to overfit).

5) Finally, a subset of 'besaetninger/bedrifter' with similar management by using similar model performance as an imperfect proxy for similar management. This subset of besaetninger/bedrifter is determined by K-fold cross validation, evaluating on a different held-out sample of the training data in each training iteration. The resulting best performing 49 besaetninger/bedrifter is then retrained and tested on a 80/20 train/validation split of the data.

1.1.3 Conclusions

- The data shows significant differences between 'besaetninger/bedrifter'. This suggests that the 'management' in the individual 'besaetninger/bedrifter' has a large impact on the results.

Data from different ‘besaetninger/bedrifter’ may therefore not be comparable, and a model including data from all ‘besaetninger/bedrifter’ may not be suitable.

- Looking at the most important features for the final model, it is found that it complies with what was hypothesized by domain experts. The most important features include:
  - besaetning ID (‘bedrift’)
  - previous hoof disorders (tidl_klovlidelse), previous digital dermatitis (prev_digital_dermatitis) and possibly correlating hoof disorders (prev_possibly_corr_klovlid)
  - where the cow is in its lactation period from its lastest calving date (days_in_lactation_period)
  - number of birthed calves (klvnr)
  - weight, milk production (ydelse) and number of milking sessions leading up to the hoof trimming date
  - race_id
  - number of hoof trimmings during the cow’s lastest ‘gold’-period (klovbeskaer_during_gold)
- Many features are highly correlated and have little to no effect on the target value. Using only a small subset of the feature set resulted in minor performance improvement.
- Hyperparameter tuning by Randomized Search Cross Validation for two hyperparameters (learning rate and max_depth) did not improve model performance.
- Training on only ‘similar’ (the best performing) ‘besaetninger/bedrifter’ greatly improves the performance.
- Some signal seems to be present in the data, especially from features of medical history, hoof trimmings during lastest ‘gold’-period, and weight measures
- Under the best of circumstances in this analysis, we obtain a model yielding a **0.83 ROC AUC score, 0.74 Precision-Recall AUC score and 0.78 accuracy score.**
- These model performance metrics may be increased by running a computationally expensive Grid Search Cross Validation across a much larger grid of hyperparameter values and on more hyperparameters.

1.1.4 Future Research

The following areas are interesting for future work:

Model - Grid Search Cross Validation across a much larger grid of hyperparameter values and on more hyperparameters. - Different models should be tested in search for better prediction performance, e.g. neural networks, random forests etc.

Data - The data contains a lot of missing values which depreciates the data quality significantly. - The weight measurements fluctuates greatly, even in the attempted corrected form. Further corrections may benefit the model. - As the data shows significant differences between ‘besaetninger/bedrifter’ due to e.g. management, alleviating these differences by alligning farmer management is likely to improve model performance. - If the model is to be used for new ‘besaetninger/bedrifter’ which are not included in the training data, it will be beneficial to not include ‘bedrift’ as a feature.

- The following additional features may be beneficial, but they require additional data:
  - **Barn (type) features.**
  - Feed and change in feed.

  - Data during the ‘gold’ period.
- Further performance enhancements may be present by splitting the data further into subsets with better data quality.

Next steps - Decision on whether to go ahead with ketosis or hoof disorder prescriptive modelling

Conda Environment In this notebook the `5517-ap1` conda environment is used. It can be installed using the `environment.yml` file along with the `requirements.txt` file.

[ ]:
```
import os
import azureml.core
from azureml.core import Workspace, Datastore, Dataset
from azureml.data.datapath import DataPath
import pandas as pd
from pathlib import Path
from pandas_profiling import ProfileReport
import holoviews as hv
import hvplot.pandas
import numpy as np

import re
import warnings
from pathlib import Path
from pprint import pprint as pp

import holoviews as hv
import hvplot.pandas
import matplotlib.pyplot as plt
import shap
from catboost import (CatBoostClassifier, EFeaturesSelectionAlgorithm,
                      EShapCalcType, Pool)
from pandas.tseries.offsets import DateOffset
from sklearn.metrics import (auc, classification_report,
                             precision_recall_curve,
                             precision_recall_fscore_support, roc_auc_score,
                             roc_curve)
from sklearn.model_selection import KFold, train_test_split
```

[ ]:
```
ws = Workspace.from_config()

# Default datastore
def_data_store = ws.get_default_datastore()

# Get the blob storage associated with the workspace
def_blob_store = Datastore(ws, "workspaceblobstore")

# Get file storage associated with the workspace
def_file_store = Datastore(ws, "workspacefilestore")
```

```
[ ]: data_folder = Path('data')
```

```
[ ]: data = Dataset.File.from_files(
         DataPath(def_blob_store,
         f'datascience-analyses/5517-datadrevet-management-ap1/*.csv'))

     data.download(
         target_path=data_folder.as_posix(), overwrite=True)
```

```
[ ]: # PARAMS
     minimum_days_of_daily_obs = 60
     interval_size = 10
     numb_of_intervals = 5
```

```
[ ]: # ------------------------------------------------
     #  Create target
     # ------------------------------------------------

     def create_target(df, unique_obs, span=5):
         klov_lidelser = ['Klovbrandbyld', 'Balleforrådnelse', 'Nydannelse',␣
      ↪'Digital dermatitis', 'Sålesår', 'Sålebløding', 'Tånekrose', 'Dobbeltsål',␣
      ↪'Hul væg, byld i hvid linje']
         targets = []
         unique_klvs = unique_obs[unique_obs.DYR_ID==df.DYR_ID.iloc[0]]['regdato']
         if len(unique_klvs)==0:
             return
         klovbrandbyld_by_landmand = ((df.navn=='Klovbrandbyld') & (df.
      ↪BEHANDLERTEKST=='Landmand'))
         for reg_dato in unique_klvs:
             horn_relaterede_three_months_prior = df[df.underkat == 'Hornrelaterede␣
      ↪klovlidelser (Sygdomme)'].regdato.between(
                 pd.to_datetime(reg_dato) - DateOffset(days=90),
                 pd.to_datetime(reg_dato) - DateOffset(days=1),
                 inclusive='both').any()
             if not horn_relaterede_three_months_prior:
                 target = df[( (~klovbrandbyld_by_landmand) & df.navn.
      ↪isin(klov_lidelser) )].regdato.between(
                     pd.to_datetime(reg_dato) - DateOffset(days=span),
                     pd.to_datetime(reg_dato) + DateOffset(days=span),
                     inclusive='both').any()
             else:
                 target = False
             dict_target = {'regdato': reg_dato, 'target': target}
             targets.append(dict_target)

         df_target = pd.DataFrame(targets)
         return df_target
```

```
#df_target = df_syg.groupby(['DYR_ID']).apply(create_target, unique_obs).
 ↪reset_index().drop(['level_1'], axis=1)
#df_target


# get robK_weight feature

def create_feature_robK_weight(df):
    reg_dato = df.regdato.iloc[0]
    first_w_dato = df[df.robK_BW.notna()].VISITDATETIME.min()

    df = df[df.VISITDATETIME.between(reg_dato -␣
 ↪DateOffset(days=interval_size*(1 + numb_of_intervals)), reg_dato,␣
 ↪inclusive='left')]

    df = df[['DYR_ID', 'regdato', 'klvdato', 'VISITDATETIME', 'robK_BW']].
 ↪resample('10D', on='VISITDATETIME', origin = reg_dato-␣
 ↪DateOffset(days=interval_size*(1 + numb_of_intervals)) ).agg(
                    {
                        'DYR_ID': 'first',
                        'regdato': 'first',
                        'klvdato': 'first',
                        'robK_BW': 'mean'
                    })
    names = {days+interval_size: str(days) + '_' + str(days+interval_size) +␣
 ↪'_days_bf_klovbskrng' for days in range(0,minimum_days_of_daily_obs,␣
 ↪interval_size)}
    df['name'] = [names[(reg_dato-idx).days] for idx in df.index]
    df.index=[0]*len(df)
    df = df.pivot(index=None, columns='name', values=['robK_BW'])
    df.columns = df.columns.map(''.join)

    return df

#df_bf_klvbeskr_weight_feature = df_hendelserprdag.
 ↪groupby(['DYR_ID','regdato']).apply(create_feature_robK_weight).
 ↪reset_index(level=2, drop=True).reset_index()

# Relative features before klovskring
def create_weight_bf_calving_relative_and_stadardized(df_bf_features_orig):
# -------------------------
#   Features bf
# -------------------------
    df_bf_features = df_bf_features_orig.copy()
    #Relative features bf
    relative_features = ['robK_BW']
    names = {days+interval_size: str(days) + '_' + str(days+interval_size) +␣
↪'_days_bf_klovbskrng' for days in range(0,minimum_days_of_daily_obs,␣
↪interval_size)}
    col_name = names[minimum_days_of_daily_obs]
    needed_columns = [name + col_name for name in relative_features]
    missing_columns = [column for column in needed_columns if column not in␣
↪df_bf_features.columns]
    df_bf_features[missing_columns] = np.nan
    bf_klv_features = []
    for feature in relative_features:
        temp_relative = df_bf_features[[col for col in df_bf_features.columns␣
↪if col.startswith(feature)]] / df_bf_features[[feature+col_name]].to_numpy()
        temp_relative.drop(feature+ col_name, axis=1, inplace=True)
        temp_relative.columns = [x + "_relative" for x in temp_relative.columns]
        bf_klv_features.append(temp_relative)
    df_bf_features_relative = pd.concat(bf_klv_features, axis=1)
    df_bf_features_full = df_bf_features.merge(df_bf_features_relative,␣
↪how='outer', right_index=True, left_index=True)
    df_bf_features_full

    # -------------------------
    #   Standardized features bf  (group deviations)
    # -------------------------
    df_bf_features = df_bf_features_orig[df_bf_features_orig.
↪columns[~df_bf_features_orig.columns.isin(['DYR_ID', 'regdato', 'klvdato',␣
↪'dato', 'klvnr', 'race_id'])]].copy()
    for feature in df_bf_features.columns:
        df_bf_features_full[feature + '_standardized'] =␣
↪(df_bf_features[feature] - df_bf_features[feature].mean()) /␣
↪df_bf_features[feature].std()

    return df_bf_features_full

#df_bf_klvbeskr_weight_feature =␣
↪create_weight_bf_calving_relative_and_stadardized(df_bf_klvbeskr_weight_feature)
#df_bf_klvbeskr_weight_feature

# -------------------------
#   Features before
# -------------------------
def create_features_before_klovskring(df):
    reg_dato = df.regdato.iloc[0]
    first_ydl_dato = df[df.ydelse.notna()].dato.min()

    df = df[df.dato.between(reg_dato - DateOffset(days=interval_size*(1 +␣
↪numb_of_intervals)), reg_dato, inclusive='left')]
```

```
    df_holds_rows = df.shape[0] > 0
    if df_holds_rows:
        save_vals = {col: df[col].iloc[0] for col in ['klvdato', 'dato',␣
↪'klvnr', 'race_id']}
        save_vals['dato'] = first_ydl_dato


    df = df[['DYR_ID', 'regdato', 'klvdato', 'klvnr', 'race_id', 'dato',␣
↪'ydelse',        'ydlfdt','ydlprt','mlkgns']].resample('10D', on='dato',␣
↪origin = reg_dato- DateOffset(days=interval_size*(1 + numb_of_intervals)) ).
↪agg(
                    {
                        'DYR_ID': 'first',
                        'regdato': 'first',
                        'ydelse': 'sum',
                        'ydlprt': 'mean',
                        'ydlfdt': 'mean',
                        'mlkgns': 'mean'
                    })
    names = {days+interval_size: str(days) + '_' + str(days+interval_size) +␣
↪'_days_bf_klovbskrng' for days in range(0,minimum_days_of_daily_obs,␣
↪interval_size)}

    df['name'] = [names[(reg_dato-idx).days] for idx in df.index]
    df.index=[0]*len(df)
    df = df.pivot(index=None, columns='name', values=['ydelse', 'ydlprt',␣
↪'ydlfdt', 'mlkgns'])
    df.columns = df.columns.map(''.join)
    if df_holds_rows:
        df = pd.concat([pd.DataFrame(save_vals, index=[0]),df], axis=1)

    return df

#df_bf_klv_features = df_dag.groupby(['DYR_ID','regdato']).
↪apply(create_features_before_klovskring).reset_index(level=2, drop=True).
↪reset_index()


# Relative features before klovskring
def create_features_bf_calving_relative_and_stadardized(df_bf_features_orig):
# -------------------------
#   Features bf
# -------------------------
    df_bf_features = df_bf_features_orig.copy()
    #Relative features bf
    relative_features = ['ydelse', 'ydlprt', 'ydlfdt', 'mlkgns']
    names = {days+interval_size: str(days) + '_' + str(days+interval_size) +␣
 ↪'_days_bf_klovbskrng' for days in range(0,minimum_days_of_daily_obs,␣
 ↪interval_size)}
    col_name = names[minimum_days_of_daily_obs - interval_size]
    needed_columns = [name + col_name for name in relative_features]
    missing_columns = [column for column in needed_columns if column not in␣
 ↪df_bf_features.columns]
    df_bf_features[missing_columns] = np.nan
    bf_klv_features = []
    for feature in relative_features:
        temp_relative = df_bf_features[[col for col in df_bf_features.columns␣
 ↪if col.startswith(feature)]] / df_bf_features[[feature+col_name]].to_numpy()
        temp_relative.drop(feature+ col_name, axis=1, inplace=True)
        temp_relative.columns = [x + "_relative" for x in temp_relative.columns]
        bf_klv_features.append(temp_relative)
    df_bf_features_relative = pd.concat(bf_klv_features, axis=1)
    df_bf_features_full = df_bf_features.merge(df_bf_features_relative,␣
 ↪how='outer', right_index=True, left_index=True)
    df_bf_features_full

    # -------------------------
    #   Standardized features bf  (group deviations)
    # -------------------------
    df_bf_features = df_bf_features_orig[df_bf_features_orig.
 ↪columns[~df_bf_features_orig.columns.isin(['DYR_ID', 'regdato', 'klvdato',␣
 ↪'dato', 'klvnr', 'race_id'])]].copy()
    for feature in df_bf_features.columns:
        df_bf_features_full[feature + '_standardized'] =␣
 ↪(df_bf_features[feature] - df_bf_features[feature].mean()) /␣
 ↪df_bf_features[feature].std()

    return df_bf_features_full

#df_bf_klv_features_full =␣
 ↪create_features_bf_calving_relative_and_stadardized(df_bf_klv_features)

#df_bf_klv_features_full['days_in_lactation_period'] = (df_bf_klv_features_full.
 ↪regdato - df_bf_klv_features_full.klvdato).dt.days
#df_bf_klv_features_full


def create_static_features(df_hendelserprdag, idx_cols):
    # 'Static' features
    df_static_features = df_hendelserprdag.groupby(idx_cols).
 ↪head(minimum_days_of_daily_obs).groupby(idx_cols).agg(
        {'bedrift':'first'}).reset_index()
    return df_static_features



# Gold period feature
def create_gold_period_length(df, df_syg):
    list_gold_period = []
    for kv_dato in df.klvdato.unique():
        last_date_span = df[df.VISITDATETIME.between(pd.to_datetime(kv_dato) -␣
 ↪DateOffset(days=150), pd.to_datetime(kv_dato) - DateOffset(days=24),␣
 ↪inclusive='left')]#.max()
        first_date = last_date_span.VISITDATETIME.min()
        last_date = last_date_span.VISITDATETIME.max()
        if pd.isnull(last_date):
            continue
        gold_period_length = (kv_dato - last_date).days
        klovbeskaer_idx = (df_syg.DYR_ID==df.DYR_ID.iloc[0]) & (df_syg.
 ↪klvdato==df.klvdato.iloc[0]) & (df_syg.navn=='Klovbeskæring')
        klovbeskaer_during_gold = df_syg[klovbeskaer_idx].regdato.
 ↪between(first_date, last_date).sum()
        dict_gold_period = {'klvdato': kv_dato, 'gold_period_length':␣
 ↪gold_period_length, "klovbeskaer_during_gold": klovbeskaer_during_gold}
        list_gold_period.append(dict_gold_period)

    df_gold_period = pd.DataFrame(list_gold_period)
    return df_gold_period

#df_gold_period = df_hendelserprdag.groupby(['DYR_ID']).
 ↪apply(create_gold_period_length, df_syg).reset_index().drop(['level_1'],␣
 ↪axis=1)
#df_gold_period

def create_klv_features(df_klv):
    # ------------------------------------------------
    #  Features from df_klv
    # ------------------------------------------------
    df_klv = df_klv[['DYR_ID', 'klvnr', 'klvdato', 'twin', 'abort',␣
 ↪'abort_ins_efter',
                     'doedfoedt', 'forloeb', 'Huldklv_dato', 'Huldklv_score',␣
 ↪'huldgold_dato', 'huldgold_score']]
    df_klv['huldklv_days_since_klv'] = (df_klv['Huldklv_dato'] -␣
 ↪df_klv['klvdato']).dt.days
    df_klv['huldgold_days_bf_klv'] = (df_klv['klvdato'] -␣
 ↪df_klv['huldgold_dato']).dt.days
    df_klv = df_klv.drop(['Huldklv_dato', 'huldgold_dato', 'klvnr'], axis=1)
    return df_klv


# ------------------------------------------------
#  Features from df_syg
# ------------------------------------------------

def create_syg_features(df, unique_obs):
    syg_features = []
    unique_klvs = unique_obs[unique_obs.DYR_ID==df.DYR_ID.iloc[0]]['regdato'] #␣
 ↪klvdato_next as these are the ones with obs. before and after klv
    if len(unique_klvs)==0:
        return
    for klovbksr_dato in unique_klvs:
        klov_lidelser = ['Klovbrandbyld', 'Balleforrådnelse', 'Nydannelse',␣
 ↪'Digital dermatitis', 'Sålesår', 'Sålebl⌀dning', 'Tånekrose', 'Dobbeltsål',␣
 ↪'Hul væg, byld i hvid linje']
        tidl_klovlidelse = len(df[(df.regdato < klovbksr_dato) & (df.navn.
 ↪isin(klov_lidelser))])

        possibly_correlating_klovlidelser = ['Sålesår', 'Sålebl⌀dning',␣
 ↪'Dobbeltsål', 'Hul væg, byld i hvid linje']
        prev_possibly_corr_klovlid = len(df[(df.regdato < klovbksr_dato) & (df.
 ↪navn.isin(possibly_correlating_klovlidelser))])

        prev_digital_dermatitis = len(df[(df.regdato < klovbksr_dato) & (df.
 ↪navn=='Digital dermatitis')])

        dict_syg_features = {'regdato': klovbksr_dato, 'tidl_klovlidelse':␣
 ↪tidl_klovlidelse, 'prev_possibly_corr_klovlid': prev_possibly_corr_klovlid,
                             'prev_digital_dermatitis': prev_digital_dermatitis}
        syg_features.append(dict_syg_features)

    df_syg_features = pd.DataFrame(syg_features)
    return df_syg_features

#df_syg_features = df_syg.groupby(['DYR_ID']).apply(create_syg_features,␣
 ↪unique_obs).reset_index().drop(['level_1'], axis=1)
#df_syg_features


# ------------------------------------------------
#  Features from df_kont
# ------------------------------------------------
def create_kont_features(df, unique_obs):
    kont_features = []
    unique_klvs = unique_obs[unique_obs.DYR_ID==df.DYR_ID.iloc[0]][['klvdato',␣
↪'regdato']]
    if len(unique_klvs)==0:
        return
    for index, row in unique_klvs.iterrows():
        date_last_kvdato = row['klvdato']
        regdato = row['regdato']
        date_after_reg = row['regdato'] + DateOffset(days=5)

        df_before_klv = df[df.kontdato.between(date_last_kvdato, regdato,␣
↪inclusive='both')].copy()
        df_before_klv['fpf'] = df_before_klv.fedtpct / df_before_klv.protpct
        df_before_klv = df_before_klv.dropna(how='all', subset=['BHB',␣
↪'acetone', 'FA_DeNovo_i_TFA', 'FA_Mixed_i_TFA', 'FA_Preformed_i_TFA']) #␣
↪Remove controls with missing values for BHB (for each calving there are a␣
↪few controls in 'goldperioden' where only fdg, mdg1, and mdg2 are measured)

        df_after_klv = df[df.kontdato.between(regdato, date_after_reg,␣
↪inclusive='right')].copy()
        df_after_klv['fpf'] = df_after_klv.fedtpct / df_after_klv.protpct
        df_after_klv = df_after_klv.dropna(how='all', subset=['BHB', 'acetone',␣
↪'FA_DeNovo_i_TFA', 'FA_Mixed_i_TFA', 'FA_Preformed_i_TFA']) # Remove␣
↪controls with missing values for BHB (for each calving there are a few␣
↪controls in 'goldperioden' where only fdg, mdg1, and mdg2 are measured)

        first_kont_bf_klv = df_before_klv[df_before_klv.kontdato ==␣
↪df_before_klv.kontdato.min()][['fpf','BHB', 'acetone',␣
↪'FA_DeNovo_i_TFA','FA_Mixed_i_TFA','FA_Preformed_i_TFA']]
        first_kont_bf_klv['days_since'] = (regdato - df_before_klv.kontdato.
↪min()).days
        first_kont_bf_klv.columns = [x + "_kont_first_bf_reg" for x in␣
↪first_kont_bf_klv.columns]

        last_kont_bf_klv = df_before_klv[df_before_klv.kontdato ==␣
↪df_before_klv.kontdato.max()][['fpf','BHB', 'acetone',␣
↪'FA_DeNovo_i_TFA','FA_Mixed_i_TFA','FA_Preformed_i_TFA']]
        last_kont_bf_klv['days_since'] = (regdato - df_before_klv.kontdato.
↪max()).days
        last_kont_bf_klv.columns = [x + "_kont_last_bf_reg" for x in␣
↪last_kont_bf_klv.columns]

        first_kont_after_klv = df_after_klv[df_after_klv.kontdato ==␣
↪df_after_klv.kontdato.min()][['fpf','BHB', 'acetone',␣
↪'FA_DeNovo_i_TFA','FA_Mixed_i_TFA','FA_Preformed_i_TFA']]
        first_kont_after_klv['days_before'] = (df_after_klv.kontdato.min() -␣
↪regdato).days
```

```
        first_kont_after_klv.columns = [x + "_kont_first_after_reg" for x in␣
↪first_kont_after_klv.columns]

        kont_id = pd.DataFrame({'regdato': regdato}, index=[0])

        df_kont_features_temp = pd.concat((first_kont_bf_klv.
↪reset_index(drop=True),last_kont_bf_klv.reset_index(drop=True),␣
↪first_kont_after_klv.reset_index(drop=True), kont_id.
↪reset_index(drop=True)), axis=1)

        kont_features.append(df_kont_features_temp)

    df_kont_features = pd.concat(kont_features, axis=0)

    return df_kont_features


#if len(df_kont)>0: # Remove when fix to kont.csv # DELETE IN THE PDF
#    df_kont_features = df_kont.groupby(['DYR_ID']).apply(create_kont_features,␣
↪unique_obs).reset_index().drop(['level_1'], axis=1)
#else:
#    df_kont_features = None
#
#df_kont_features
```

[ ]:
```
def create_dataset(df_hendelserprdag, df_syg, df_kont, df_klv, df_dag):

    # Idx cols
    idx_cols = ['DYR_ID', 'regdato']
    df_syg = df_syg.dropna(subset=idx_cols)
    # Sort by date
    df_syg = df_syg.sort_values(by=idx_cols + ['klvdato'])

    from_date = pd.to_datetime('2019-01-01')

    df_syg_klovbskr = df_syg[(df_syg.regdato >= from_date) & (df_syg.navn ==␣
↪'Klovbeskæring')]

    # get klvdato_next
    # df_klv_next = df_dag[['DYR_ID','klvnr','klvdato']].drop_duplicates().
↪rename(columns={'klvdato': 'klvdato_next'})
    # df_klv_next['klvnr'] -=1
    # df_dag = df_dag.merge(df_klv_next[['DYR_ID', 'klvnr','klvdato_next']],
    #                                                 how = 'left',␣
↪on=['DYR_ID','klvnr'])
    ##
    df_dag = df_dag.merge(df_syg_klovbskr[['DYR_ID', 'regdato','klvdato']],
                                                        how = 'left',␣
↪on=['DYR_ID','klvdato'])

    df_hendelserprdag = df_hendelserprdag.merge(df_syg_klovbskr[['DYR_ID',␣
↪'regdato','klvdato']],
                                                        how = 'left',␣
↪on=['DYR_ID','klvdato'])

    # look only at dyr-regdato pairs that have certain length from klvdato to␣
↪regdato (ensures enough data prior to regdato)
    days_between_regdato_and_klvdato = (df_dag.regdato - df_dag.klvdato).dt.days
    df_dag = df_dag[(days_between_regdato_and_klvdato >␣
↪minimum_days_of_daily_obs)]

    df_dag = df_dag.rename(columns={'ydl': 'ydelse'})

    # Preliminary setting
    unique_obs = df_dag[['DYR_ID', 'regdato', 'klvdato']].drop_duplicates().
↪dropna()
    unique_obs

    if len(unique_obs)==0:
        return

    # -------------------------
    #   Features before klovskr
    # -------------------------
    df_bf_klvbeskr_weight_feature = df_hendelserprdag.
↪groupby(['DYR_ID','regdato']).apply(create_feature_robK_weight).
↪reset_index(level=2, drop=True).reset_index()
    df_bf_klvbeskr_weight_feature =␣
↪create_weight_bf_calving_relative_and_stadardized(df_bf_klvbeskr_weight_feature)

    df_bf_klv_features = df_dag.groupby(['DYR_ID','regdato']).
↪apply(create_features_before_klovskring).reset_index(level=2, drop=True).
↪reset_index()
    df_bf_klv_features_full =␣
↪create_features_bf_calving_relative_and_stadardized(df_bf_klv_features)
    df_bf_klv_features_full['days_in_lactation_period'] =␣
↪(df_bf_klv_features_full.regdato - df_bf_klv_features_full.klvdato).dt.days


    # 'Static' features
    df_static_features = create_static_features(df_hendelserprdag, idx_cols)
```

```
# Gold period feature
df_gold_period = df_hendelserprdag.groupby(['DYR_ID']).
↪apply(create_gold_period_length, df_syg).reset_index().drop(['level_1'],␣
↪axis=1)

# ------------------------------------------------
#  Features from df_klv
# ------------------------------------------------
df_klv = create_klv_features(df_klv)

# ------------------------------------------------
#  Features from df_syg
# ------------------------------------------------
df_syg_features = df_syg.groupby(['DYR_ID']).apply(create_syg_features,␣
↪unique_obs).reset_index().drop(['level_1'], axis=1)

# ------------------------------------------------
#  Features from df_kont
# ------------------------------------------------
if len(df_kont)>0: # Remove when fix to kont.csv # DELETE IN THE PDF
    df_kont_features = df_kont.groupby(['DYR_ID']).
↪apply(create_kont_features, unique_obs).reset_index().drop(['level_1'],␣
↪axis=1)
else:
    df_kont_features = None

# ------------------------------------------------
#  Merge features
# ------------------------------------------------
df_full_features = df_bf_klv_features_full
df_full_features = df_full_features.merge(
    df_bf_klvbeskr_weight_feature, how='left', on=idx_cols)
df_full_features = df_full_features.merge(
    df_static_features, how='left', on=idx_cols)
df_full_features = df_full_features.merge(
    df_klv, how='left', on=['DYR_ID', 'klvdato'])
df_full_features = df_full_features.merge(
    df_syg_features, how='left', on=idx_cols)
df_full_features = df_full_features.merge(
    df_gold_period, how='left', on=['DYR_ID', 'klvdato'])
if df_kont_features is not None: # Remove when fix to kont.csv # DELETE IN␣
↪THE PDF
    df_full_features = df_full_features.merge(
        df_kont_features, how='left', on=idx_cols)

df_full_features
```

```
    # ------------------------------------------------
    #  Create final additional features
    # ------------------------------------------------

    df_full_features['month_of_year'] = df_full_features['klvdato'].dt.month.
↪astype(str)

    # ------------------------------------------------
    #  Create target
    # ------------------------------------------------
    df_target = df_syg.groupby(['DYR_ID']).apply(create_target, unique_obs).
↪reset_index().drop(['level_1'], axis=1)

    # ------------------------------------------------
    #  Merge target and features
    # ------------------------------------------------
    df_final = df_full_features.merge(df_target, how='left', on=idx_cols)
    df_final['target'] = df_final['target'] == True

    return df_final
```

```
[ ]: # selected 59 besaetninger with over 200 unique cows, and previously many hoof␣
     ↪disorder registrations
     candidate_bedrifter = pd.Index(pd.read_json('candidate_bedrifter.
     ↪json')['candidate_bedrifter'].tolist(), dtype='int64')
     candidate_bedrifter.shape
```

```
[ ]: (59,)
```

```
[ ]: from tqdm import tqdm
     final_dfs = []
     for idx, bedrift in enumerate(tqdm(candidate_bedrifter)):
         if bedrift == 'OBFUSCATED': # DELETE IN THE PDF
             continue

         # Load data
         df_hendelserprdag = pd.read_csv(data_folder/f'hendelserprdag{bedrift}.csv')
         df_hendelserprdag = df_hendelserprdag[['DYR_ID', 'klvdato', 'klvnr',␣
     ↪'race_id',
                                                 'VISITDATETIME',␣
     ↪'Ydelse_total','MILKFAT', 'MILKPROTEIN',
                                               'robK_BW','MILKSPEEDMAX',␣
     ↪'MILKTEMPERATURE', 'MILKTIMEKO',
                                                 'LFDEADMILKTIME',␣
     ↪'RFDEADMILKTIME', 'LRDEADMILKTIME', 'RRDEADMILKTIME']]
```

```
    df_hendelserprdag = df_hendelserprdag.assign(bedrift = bedrift)
    df_hendelserprdag.klvdato = pd.to_datetime(df_hendelserprdag.klvdato,␣
↪format='%d/%m/%y')
    df_hendelserprdag.VISITDATETIME = pd.to_datetime(df_hendelserprdag.
↪VISITDATETIME, format='%d%b%Y:%H:%M:%S')


    df_dag = pd.read_csv(data_folder/f'dag{bedrift}.csv', encoding='latin1')
    df_dag = df_dag.rename(columns={'dyr_id':'DYR_ID'})
    df_dag.klvdato = pd.to_datetime(df_dag.klvdato, format='%d/%m/%y')
    df_dag.dato = pd.to_datetime(df_dag.dato, format='%d/%m/%y')
    df_dag = df_dag[['DYR_ID','klvdato','klvnr','race_id','dato','ydl',␣
↪'ydlfdt','ydlprt','mlkgns']]

    df_syg = pd.read_csv(data_folder/f'syg{bedrift}.csv', encoding='latin1')
    df_syg = df_syg.rename(columns={'Dyr_Id':'DYR_ID'})
    df_syg.regdato = pd.to_datetime(df_syg.regdato, format='%d%b%y')
    df_syg.klvdato = pd.to_datetime(df_syg.klvdato, format='%d%b%y')
    df_syg = df_syg[['DYR_ID', 'regdato','klvdato', 'navn', 'overkat',␣
↪'underkat', 'type', 'BEHANDLERTEKST']]

    df_kont = pd.read_csv(data_folder/f'kont{bedrift}.csv')
    df_kont = df_kont.rename(columns={'dyr_id':'DYR_ID'})
    df_kont = df_kont[['DYR_ID', 'kontdato', 'fedtpct', 'protpct', 'BHB',␣
↪'acetone', 'FA_DeNovo_i_TFA','FA_Mixed_i_TFA', 'FA_Preformed_i_TFA']]
    df_kont.kontdato = pd.to_datetime(df_kont.kontdato, format='%d%b%y')

    df_klv = pd.read_csv(data_folder/f'Kaelvninger{bedrift}.csv',␣
↪encoding='latin1')
    df_klv = df_klv.rename(columns={'dyr_id':'DYR_ID'})
    df_klv = df_klv[['DYR_ID', 'klvnr', 'klvdato', 'twin', 'abort',␣
↪'abort_ins_efter',
                     'doedfoedt', 'forloeb', 'Huldklv_dato', 'Huldklv_score',␣
↪'huldgold_dato', 'huldgold_score']]
    df_klv.klvdato = pd.to_datetime(df_klv.klvdato, format='%d/%m/%y')
    df_klv.Huldklv_dato = pd.to_datetime(df_klv.Huldklv_dato, format='%d%b%y')
    df_klv.huldgold_dato = pd.to_datetime(df_klv.huldgold_dato, format='%d%b%y')

    # Create dataset
    df_final = create_dataset(df_hendelserprdag, df_syg, df_kont, df_klv,␣
↪df_dag)

    final_dfs.append(df_final)
```

```
  7%|          | 4/59 [01:50<24:29, 26.73s/it]Columns (16) have mixed types.
Specify dtype option on import or set low_memory=False.
 12%|          | 7/59 [04:10<36:43, 42.37s/it]Columns (16) have mixed types.
```

```
Specify dtype option on import or set low_memory=False.
 20%|         | 12/59 [07:11<26:07, 33.35s/it]Columns (16) have mixed types.
Specify dtype option on import or set low_memory=False.
 69%|       | 41/59 [25:28<11:01, 36.77s/it]Columns (16) have mixed types.
Specify dtype option on import or set low_memory=False.
 88%|      | 52/59 [32:45<05:08, 44.10s/it]Columns (16) have mixed types.
Specify dtype option on import or set low_memory=False.
100%|      | 59/59 [37:15<00:00, 37.89s/it]
```

[ ]:
```
df_final_full = pd.concat(final_dfs)
print(df_hendelserprdag.shape[0], df_syg.shape[0], df_kont.shape[0], df_klv.
 ↪shape[0])
```

```
415516 5976 10407 734
```

1.1.5 Training model on 59 besaetninger with high sickness and treatment registrations

[ ]:
```
df_final_full.to_csv(data_folder/'df_final_full_klovbeskr_many.csv', index =␣
 ↪False)
df_final_full.info()
```

[ ]:
```
df_full_dataset = pd.read_csv(data_folder/'df_final_full_klovbeskr_many.csv')
df_full_dataset = df_full_dataset.drop(['DYR_ID','klvdato',␣
 ↪'regdato',         'klvdato',         'dato'], axis=1)
df_full_dataset['bedrift'] = df_full_dataset['bedrift'].astype(str)

X = df_full_dataset.drop(["target"], axis=1)
y = df_full_dataset["target"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=5)

print(f'Length of test dataset: {len(y_test)}')
print(f'Percentage of klov cases: {round(len(y_test[y_test==1]) / len(y_test),␣
 ↪2)}')
```

```
Length of test dataset: 15573
Percentage of klov cases: 0.46
```

[ ]:
```
cat_col_indices = np.where(X_train.dtypes == object)[0]
model = CatBoostClassifier(
    #class_weights=(1,10),
    n_estimators=20000
)

model.fit(
    X_train,
    y_train,
    eval_set=(X_test, y_test),
    verbose=False,
    cat_features=cat_col_indices,
    early_stopping_rounds=100
)
```

```
[ ]: <catboost.core.CatBoostClassifier at 0x7fb031073880>
```

```
[ ]: y_pred_prob = model.predict_proba(X_test, )[:,1]
     y_pred = y_pred_prob >= 0.5
```

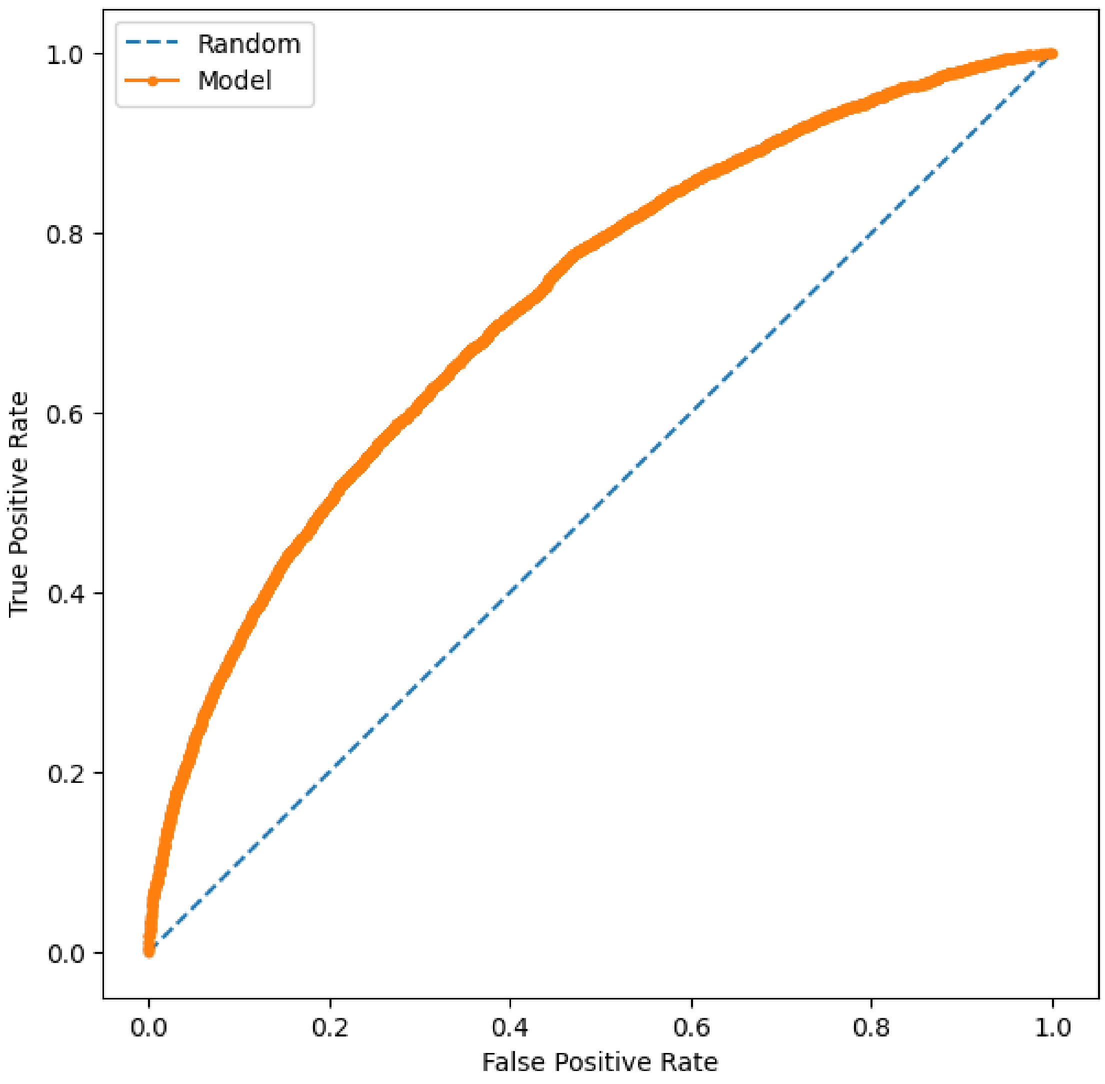
```
[ ]: pp(classification_report(y_true=y_test.values,y_pred=y_pred))
```

```
('              precision    recall  f1-score   support\n'
 '\n'
 '       False       0.67      0.74      0.70      8394\n'
 '        True       0.65      0.57      0.61      7179\n'
 '\n'
 '    accuracy                           0.66     15573\n'
 '   macro avg       0.66      0.66      0.66     15573\n'
 'weighted avg       0.66      0.66      0.66     15573\n')
```

```
[ ]: # Area under ROC curve
     roc_auc_score(y_score=y_pred_prob, y_true=y_test)
```

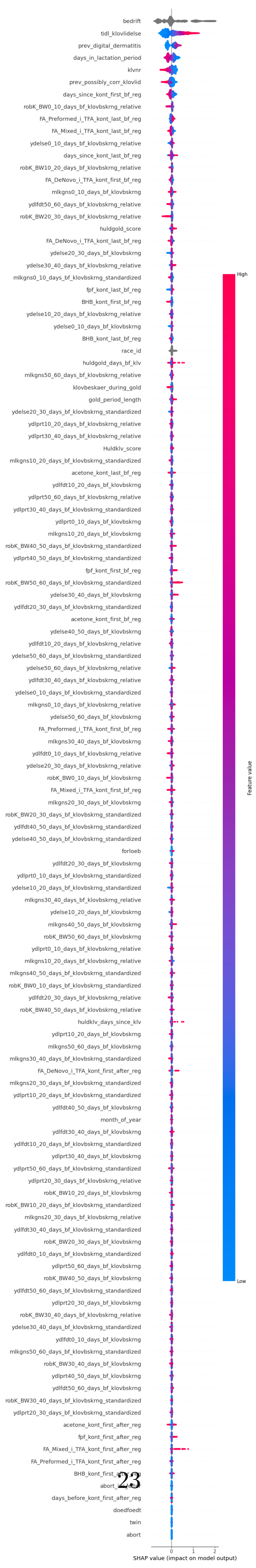
```
[ ]: 0.7214166202266472
```

```
[ ]: # Plot ROC curve
     random_probs = [0 for _ in range(len(y_test))]
     model_fpr, model_tpr, _ = roc_curve(y_test, y_pred_prob)
     random_fpr, random_tpr, _ = roc_curve(y_test, random_probs)
     plt.plot(random_fpr, random_tpr, linestyle='--', label='Random')
     plt.plot(model_fpr, model_tpr, marker='.', label='Model')
     plt.xlabel('False Positive Rate')
     plt.ylabel('True Positive Rate')
     plt.legend()
     plt.rcParams["figure.figsize"] = (7,7)
     plt.show()
```



```
[ ]: shap_values = model.get_feature_importance(Pool(X_test,␣
      ↪label=y_test,cat_features=cat_col_indices),
                                                                  ␣
      ↪type="ShapValues")
     shap.summary_plot(shap_values[:,:-1], X_test, max_display=1000)
```

```
No data for colormapping provided via 'c'. Parameters 'vmin', 'vmax' will be
ignored
```


bedrift
tidl_klovlidelse
prev_digital_dermatitis
days_in_lactation_period
klvnr
prev_possibly_corr_klovlid
days_since_kont_first_bf_reg
robK_BW0_10_days_bf_klovbskrng_relative
FA_Preformed_i_TFA_kont_last_bf_reg
FA_Mixed_i_TFA_kont_last_bf_reg
ydelse0_10_days_bf_klovbskrng_relative
days_since_kont_last_bf_reg
robK_BW10_20_days_bf_klovbskrng_relative
FA_DeNovo_i_TFA_kont_first_bf_reg
mlkgns0_10_days_bf_klovbskrng
ydlfdt50_60_days_bf_klovbskrng_relative
robK_BW20_30_days_bf_klovbskrng_relative
huldgold_score
FA_DeNovo_i_TFA_kont_last_bf_reg
ydelse20_30_days_bf_klovbskrng
ydelse30_40_days_bf_klovbskrng_relative
mlkgns0_10_days_bf_klovbskrng_standardized
fpf_kont_last_bf_reg
BHB_kont_first_bf_reg
ydelse10_20_days_bf_klovbskrng_relative
ydelse0_10_days_bf_klovbskrng
BHB_kont_last_bf_reg
race_id
huldgold_days_bf_klv
mlkgns50_60_days_bf_klovbskrng_relative
klovbeskaer_during_gold
gold_period_length
ydelse20_30_days_bf_klovbskrng_standardized
ydlprt10_20_days_bf_klovbskrng_relative
ydlprt30_40_days_bf_klovbskrng_relative
Huldklv_score
mlkgns10_20_days_bf_klovbskrng_standardized
acetone_kont_last_bf_reg
ydlfdt10_20_days_bf_klovbskrng
ydlprt50_60_days_bf_klovbskrng_relative
ydlprt30_40_days_bf_klovbskrng_standardized
ydlprt0_10_days_bf_klovbskrng
mlkgns10_20_days_bf_klovbskrng
robK_BW40_50_days_bf_klovbskrng_standardized
ydlprt40_50_days_bf_klovbskrng_standardized
fpf_kont_first_bf_reg
robK_BW50_60_days_bf_klovbskrng_standardized
ydelse30_40_days_bf_klovbskrng
ydlfdt20_30_days_bf_klovbskrng_standardized
acetone_kont_first_bf_reg
ydelse40_50_days_bf_klovbskrng
ydlfdt10_20_days_bf_klovbskrng_relative
ydelse50_60_days_bf_klovbskrng_standardized
ydelse50_60_days_bf_klovbskrng_relative
ydlfdt30_40_days_bf_klovbskrng_relative
ydelse0_10_days_bf_klovbskrng_standardized
mlkgns0_10_days_bf_klovbskrng_relative
ydelse50_60_days_bf_klovbskrng
FA_Preformed_i_TFA_kont_first_bf_reg
mlkgns30_40_days_bf_klovbskrng
ydlfdt0_10_days_bf_klovbskrng_relative
ydelse20_30_days_bf_klovbskrng_relative
robK_BW0_10_days_bf_klovbskrng
FA_Mixed_i_TFA_kont_first_bf_reg
mlkgns20_30_days_bf_klovbskrng
robK_BW20_30_days_bf_klovbskrng_standardized
ydlfdt40_50_days_bf_klovbskrng_standardized
ydelse40_50_days_bf_klovbskrng_standardized
forloeb
ydlfdt20_30_days_bf_klovbskrng
ydlprt0_10_days_bf_klovbskrng_standardized
ydelse10_20_days_bf_klovbskrng_standardized
mlkgns30_40_days_bf_klovbskrng_relative
ydelse10_20_days_bf_klovbskrng
mlkgns40_50_days_bf_klovbskrng
robK_BW50_60_days_bf_klovbskrng
ydlprt0_10_days_bf_klovbskrng_relative
mlkgns10_20_days_bf_klovbskrng_relative
mlkgns40_50_days_bf_klovbskrng_standardized
robK_BW0_10_days_bf_klovbskrng_standardized
ydlfdt20_30_days_bf_klovbskrng_relative
robK_BW40_50_days_bf_klovbskrng_relative
huldklv_days_since_klv
ydlprt10_20_days_bf_klovbskrng
mlkgns50_60_days_bf_klovbskrng
mlkgns30_40_days_bf_klovbskrng_standardized
FA_DeNovo_i_TFA_kont_first_after_reg
mlkgns20_30_days_bf_klovbskrng_standardized
ydlprt10_20_days_bf_klovbskrng_standardized
ydlfdt40_50_days_bf_klovbskrng
month_of_year
ydlfdt30_40_days_bf_klovbskrng
ydlfdt10_20_days_bf_klovbskrng_standardized
ydlprt30_40_days_bf_klovbskrng
ydlprt50_60_days_bf_klovbskrng_standardized
ydlprt20_30_days_bf_klovbskrng_relative
robK_BW10_20_days_bf_klovbskrng
robK_BW10_20_days_bf_klovbskrng_standardized
mlkgns20_30_days_bf_klovbskrng_relative
ydlfdt30_40_days_bf_klovbskrng_standardized
robK_BW20_30_days_bf_klovbskrng
ydlfdt0_10_days_bf_klovbskrng_standardized
ydlprt50_60_days_bf_klovbskrng
robK_BW40_50_days_bf_klovbskrng
ydlfdt50_60_days_bf_klovbskrng_standardized
ydlprt20_30_days_bf_klovbskrng
robK_BW30_40_days_bf_klovbskrng_relative
ydelse30_40_days_bf_klovbskrng_standardized
ydlfdt0_10_days_bf_klovbskrng
mlkgns50_60_days_bf_klovbskrng_standardized
robK_BW30_40_days_bf_klovbskrng
ydlprt40_50_days_bf_klovbskrng
ydlfdt50_60_days_bf_klovbskrng
robK_BW30_40_days_bf_klovbskrng_standardized
ydlprt20_30_days_bf_klovbskrng_standardized
acetone_kont_first_after_reg
fpf_kont_first_after_reg
FA_Mixed_i_TFA_kont_first_after_reg
FA_Preformed_i_TFA_kont_first_after_reg
BHB_kont_first_after_reg
abort_
days_before_kont_first_after_reg
doedfoedt
twin
abort
0
1
2
SHAP value (impact on model output)
High
Low
Feature value

1.1.6 Training model on all eligible besaetninger

```
[ ]: hendelserprdag_files = [i for i in data_folder.glob('hendelserprdag*.csv')]
     bedrifts = [re.findall('\d{2,}', str(file))[0] for file in hendelserprdag_files]
```

```
[ ]: len(bedrifts)
```

```
[ ]: 357
```

```
[ ]: # the besaetninger that were not included lacked insufficient data to create␣
      ↪the neccessary dataset for modeling.
     # these besaetninger resulted in numerous errors  during the dataset creation␣
      ↪done inside a notebook running
     # on a cloud compute instance over several hours.
     # The dataset is loaded here:
     df_full_dataset = pd.read_csv(data_folder/'df_final_full_klovbeskr_all.csv')
     df_full_dataset = df_full_dataset.drop(['DYR_ID','klvdato',␣
      ↪'regdato',        'klvdato',        'dato'], axis=1)
     df_full_dataset['bedrift'] = df_full_dataset['bedrift'].astype(str)
```

```
[ ]: X = df_full_dataset.drop(["target"], axis=1)
     y = df_full_dataset["target"]

     X_train, X_test, y_train, y_test = train_test_split(
         X, y, test_size=0.25, random_state=5)

     print(f'Length of test dataset: {len(y_test)}')
     print(f'Percentage of klov cases: {round(len(y_test[y_test==1]) / len(y_test),␣
      ↪2)}')
```

```
Length of test dataset: 61341
Percentage of klov cases: 0.33
```

```
[ ]: cat_col_indices = np.where(X_train.dtypes == object)[0]
     model = CatBoostClassifier(
         #class_weights=(1,10),
         n_estimators=20000
     )

     model.fit(
         X_train,
         y_train,
         eval_set=(X_test, y_test),
         verbose=False,
         cat_features=cat_col_indices,
         early_stopping_rounds=100
     )
```

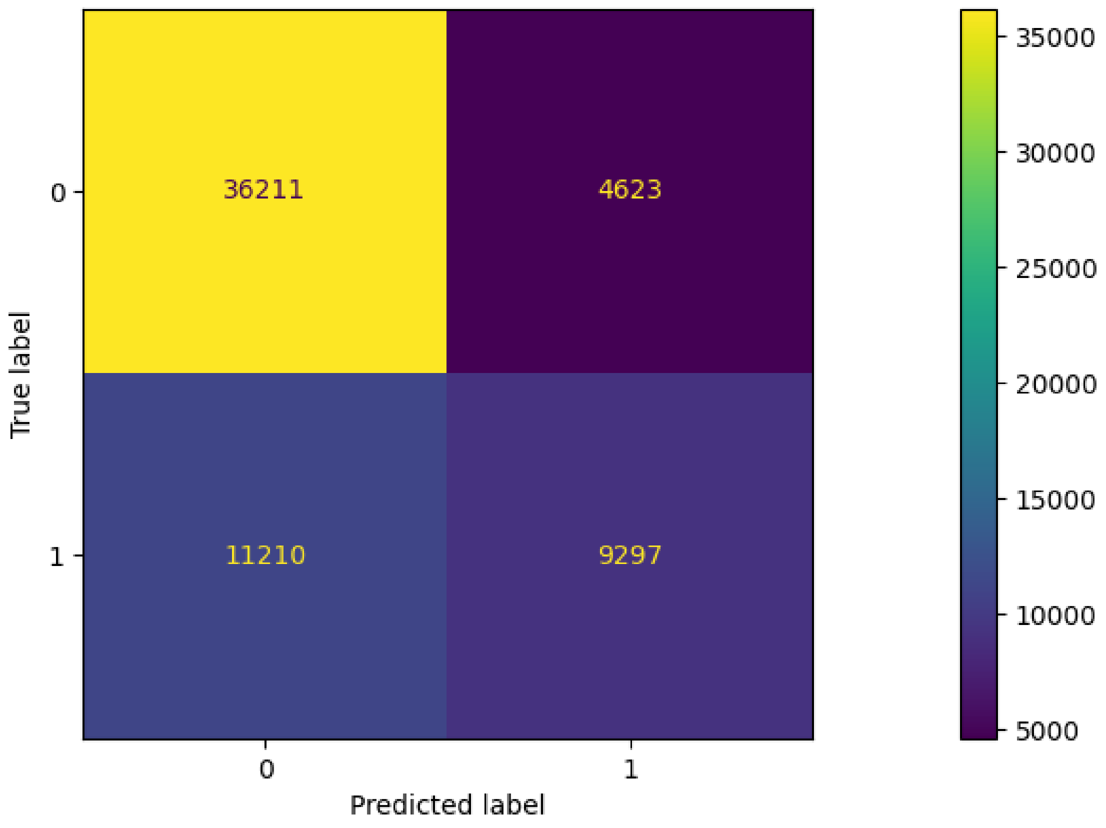
```
[ ]: <catboost.core.CatBoostClassifier at 0x7fb030c48b50>
```

```
[ ]: y_pred_prob = model.predict_proba(X_test, )[:,1]
     y_pred = y_pred_prob >= 0.5
```

```
[ ]: from sklearn.metrics import confusion_matrix
     from sklearn.metrics import ConfusionMatrixDisplay
     cm = confusion_matrix(y_true=y_test.values,y_pred=y_pred)
     pp(cm)
     %matplotlib inline
     ConfusionMatrixDisplay(cm).plot()
```

```
array([[36211,  4623],
       [11210,  9297]])
```

```
[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
     0x7fada55e4a00>
```



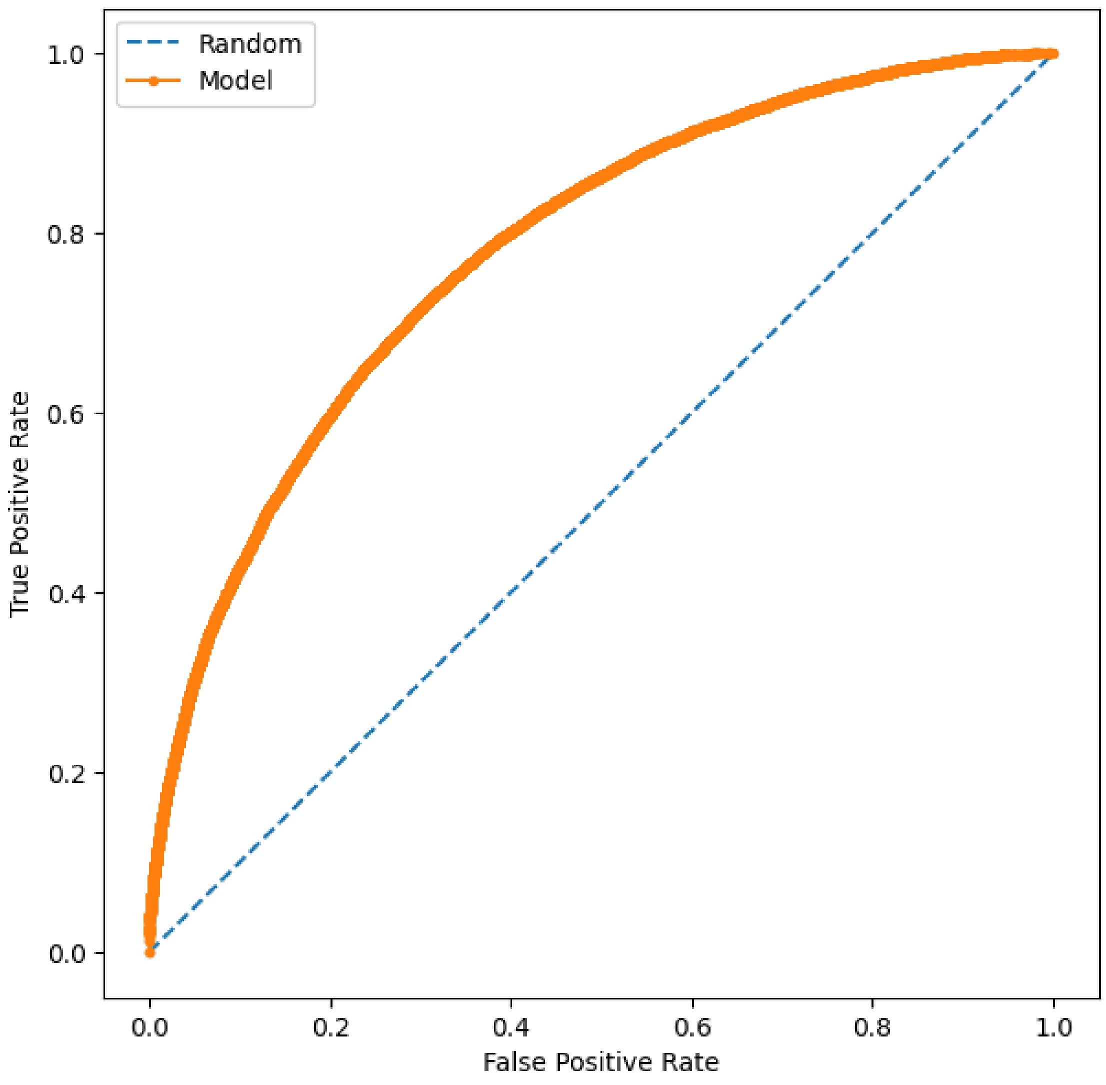
```
[ ]: pp(classification_report(y_true=y_test.values,y_pred=y_pred))
```

```
('               precision    recall  f1-score   support\n'
 '\n'
 '       False       0.76      0.89      0.82     40834\n'
 '        True       0.67      0.45      0.54     20507\n'
 '\n'
 '    accuracy                           0.74     61341\n'
 '   macro avg       0.72      0.67      0.68     61341\n'
 'weighted avg       0.73      0.74      0.73     61341\n')
```

```
[ ]: # Area under ROC curve
     roc_auc_score(y_score=y_pred_prob, y_true=y_test)
```

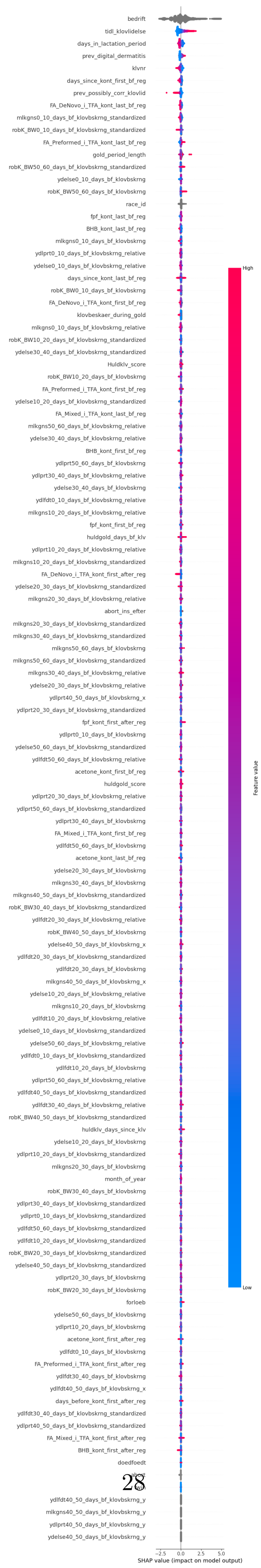
```
[ ]: 0.7818724892472659
```

```
[ ]: # Plot ROC curve
     random_probs = [0 for _ in range(len(y_test))]
     model_fpr, model_tpr, _ = roc_curve(y_test, y_pred_prob)
     random_fpr, random_tpr, _ = roc_curve(y_test, random_probs)
     plt.plot(random_fpr, random_tpr, linestyle='--', label='Random')
     plt.plot(model_fpr, model_tpr, marker='.', label='Model')
     plt.xlabel('False Positive Rate')
     plt.ylabel('True Positive Rate')
     plt.legend()
     plt.rcParams["figure.figsize"] = (7,7)
     plt.show()
```



```
[ ]: shap_values = model.get_feature_importance(Pool(X_test,␣
      ↪label=y_test,cat_features=cat_col_indices),
                                                                        ␣
      ↪type="ShapValues")
     shap.summary_plot(shap_values[:,:-1], X_test, max_display=1000)
```

```
All-NaN slice encountered
No data for colormapping provided via 'c'. Parameters 'vmin', 'vmax' will be
ignored
```

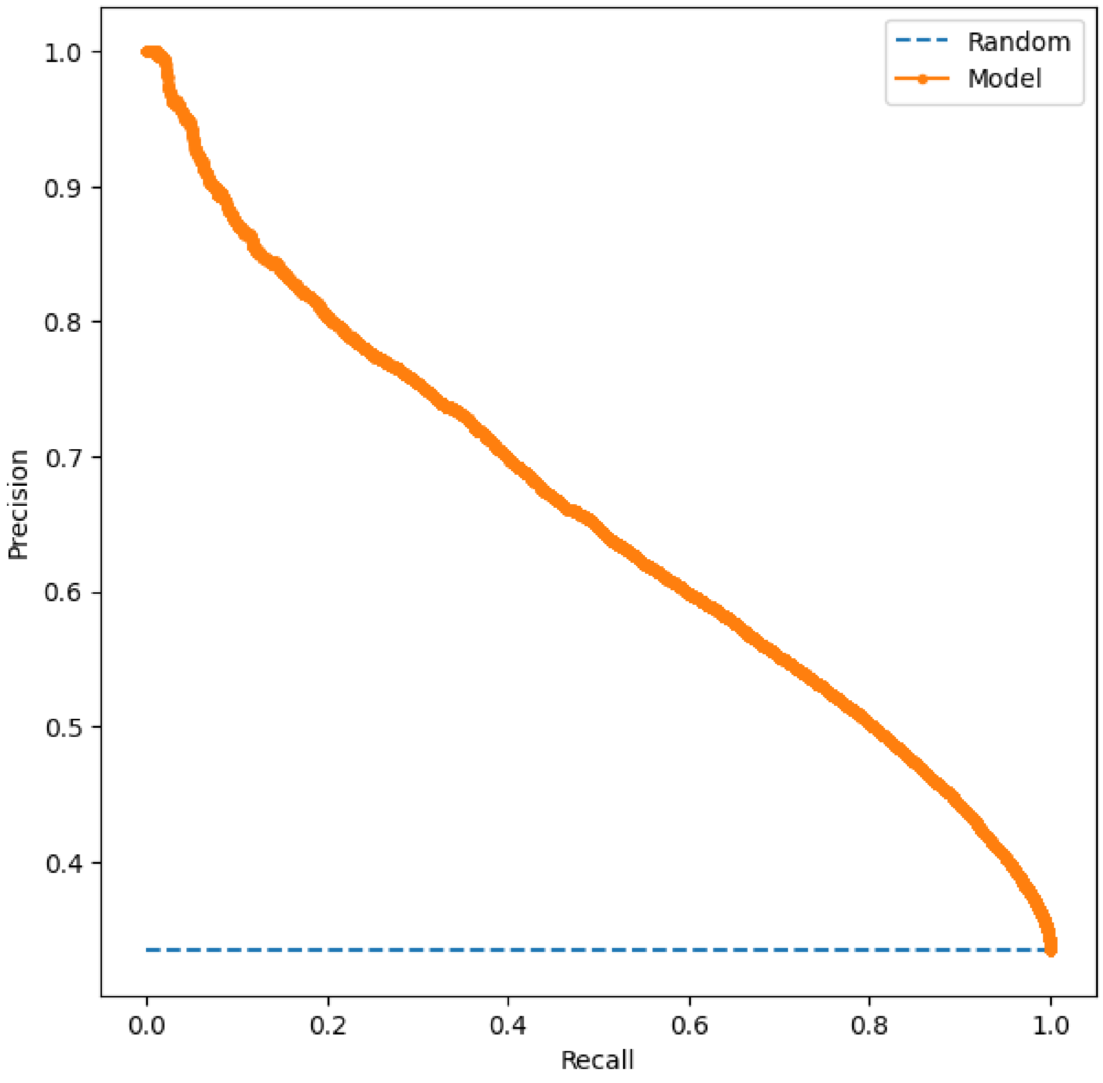

bedrift
tidl_klovlidelse
days_in_lactation_period
prev_digital_dermatitis
klvnr
days_since_kont_first_bf_reg
prev_possibly_corr_klovlid
FA_DeNovo_i_TFA_kont_last_bf_reg
mlkgns0_10_days_bf_klovbskrng_standardized
robK_BW0_10_days_bf_klovbskrng_standardized
FA_Preformed_i_TFA_kont_last_bf_reg
gold_period_length
robK_BW50_60_days_bf_klovbskrng_standardized
ydelse0_10_days_bf_klovbskrng
robK_BW50_60_days_bf_klovbskrng
race_id
fpf_kont_last_bf_reg
BHB_kont_last_bf_reg
mlkgns0_10_days_bf_klovbskrng
ydlprt0_10_days_bf_klovbskrng_relative
ydelse0_10_days_bf_klovbskrng_relative
days_since_kont_last_bf_reg
robK_BW0_10_days_bf_klovbskrng
FA_DeNovo_i_TFA_kont_first_bf_reg
klovbeskaer_during_gold
mlkgns0_10_days_bf_klovbskrng_relative
robK_BW10_20_days_bf_klovbskrng_standardized
ydelse30_40_days_bf_klovbskrng_standardized
Huldklv_score
robK_BW10_20_days_bf_klovbskrng
FA_Preformed_i_TFA_kont_first_bf_reg
ydelse10_20_days_bf_klovbskrng_standardized
FA_Mixed_i_TFA_kont_last_bf_reg
mlkgns50_60_days_bf_klovbskrng_relative
ydelse30_40_days_bf_klovbskrng_relative
BHB_kont_first_bf_reg
ydlprt50_60_days_bf_klovbskrng
ydlprt30_40_days_bf_klovbskrng_relative
ydelse30_40_days_bf_klovbskrng
ydlfdt0_10_days_bf_klovbskrng_relative
mlkgns10_20_days_bf_klovbskrng_relative
fpf_kont_first_bf_reg
huldgold_days_bf_klv
ydlprt10_20_days_bf_klovbskrng_relative
mlkgns10_20_days_bf_klovbskrng_standardized
FA_DeNovo_i_TFA_kont_first_after_reg
ydelse20_30_days_bf_klovbskrng_standardized
mlkgns20_30_days_bf_klovbskrng_relative
abort_ins_efter
mlkgns20_30_days_bf_klovbskrng_standardized
mlkgns30_40_days_bf_klovbskrng_standardized
mlkgns50_60_days_bf_klovbskrng
mlkgns50_60_days_bf_klovbskrng_standardized
mlkgns30_40_days_bf_klovbskrng_relative
ydelse20_30_days_bf_klovbskrng_relative
ydlprt40_50_days_bf_klovbskrng_x
ydlprt20_30_days_bf_klovbskrng_standardized
fpf_kont_first_after_reg
ydlprt0_10_days_bf_klovbskrng
ydelse50_60_days_bf_klovbskrng_standardized
ydlfdt50_60_days_bf_klovbskrng_relative
acetone_kont_first_bf_reg
huldgold_score
ydlprt20_30_days_bf_klovbskrng_relative
ydlprt50_60_days_bf_klovbskrng_standardized
ydlprt30_40_days_bf_klovbskrng
FA_Mixed_i_TFA_kont_first_bf_reg
ydlfdt50_60_days_bf_klovbskrng
acetone_kont_last_bf_reg
ydelse20_30_days_bf_klovbskrng
mlkgns30_40_days_bf_klovbskrng
mlkgns40_50_days_bf_klovbskrng_standardized
robK_BW30_40_days_bf_klovbskrng_standardized
ydlfdt20_30_days_bf_klovbskrng_relative
robK_BW40_50_days_bf_klovbskrng
ydelse40_50_days_bf_klovbskrng_x
ydlfdt20_30_days_bf_klovbskrng_standardized
ydlfdt20_30_days_bf_klovbskrng
mlkgns40_50_days_bf_klovbskrng_x
ydelse10_20_days_bf_klovbskrng_relative
mlkgns10_20_days_bf_klovbskrng
ydlfdt10_20_days_bf_klovbskrng_relative
ydelse0_10_days_bf_klovbskrng_standardized
ydelse50_60_days_bf_klovbskrng_relative
ydlfdt0_10_days_bf_klovbskrng_standardized
ydlfdt10_20_days_bf_klovbskrng
ydlprt50_60_days_bf_klovbskrng_relative
ydlfdt40_50_days_bf_klovbskrng_standardized
ydlfdt30_40_days_bf_klovbskrng_relative
robK_BW40_50_days_bf_klovbskrng_standardized
huldklv_days_since_klv
ydelse10_20_days_bf_klovbskrng
ydlprt10_20_days_bf_klovbskrng_standardized
mlkgns20_30_days_bf_klovbskrng
month_of_year
robK_BW30_40_days_bf_klovbskrng
ydlprt30_40_days_bf_klovbskrng_standardized
ydlprt0_10_days_bf_klovbskrng_standardized
ydlfdt50_60_days_bf_klovbskrng_standardized
ydlfdt10_20_days_bf_klovbskrng_standardized
robK_BW20_30_days_bf_klovbskrng_standardized
ydelse40_50_days_bf_klovbskrng_standardized
ydlprt20_30_days_bf_klovbskrng
robK_BW20_30_days_bf_klovbskrng
forloeb
ydelse50_60_days_bf_klovbskrng
ydlprt10_20_days_bf_klovbskrng
acetone_kont_first_after_reg
ydlfdt0_10_days_bf_klovbskrng
FA_Preformed_i_TFA_kont_first_after_reg
ydlfdt30_40_days_bf_klovbskrng
ydlfdt40_50_days_bf_klovbskrng_x
days_before_kont_first_after_reg
ydlfdt30_40_days_bf_klovbskrng_standardized
ydlprt40_50_days_bf_klovbskrng_standardized
FA_Mixed_i_TFA_kont_first_after_reg
BHB_kont_first_after_reg
doedfoedt
abort
twin
ydlfdt40_50_days_bf_klovbskrng_y
mlkgns40_50_days_bf_klovbskrng_y
ydlprt40_50_days_bf_klovbskrng_y
ydelse40_50_days_bf_klovbskrng_y
−2.5
0.0
2.5
5.0
SHAP value (impact on model output)
High
Low
Feature value

1.1.7 Precision-Recall Curve

```
[ ]: # Precision-recall curve
     precision, recall, thresholds = precision_recall_curve(y_test, y_pred_prob)
     # Area under precision-recall curve
     auc(recall, precision)
```

```
[ ]: 0.6551764564803368
```

```
[ ]: # Plot precision-recall curve
     random = len(y_test[y_test==1]) / len(y_test)
     plt.plot([0, 1], [random, random], linestyle='--', label='Random')
     plt.plot(recall, precision, marker='.', label='Model')
     plt.xlabel('Recall')
     plt.ylabel('Precision')
     plt.legend()
     plt.rcParams["figure.figsize"] = (7,7)
     plt.show()
```

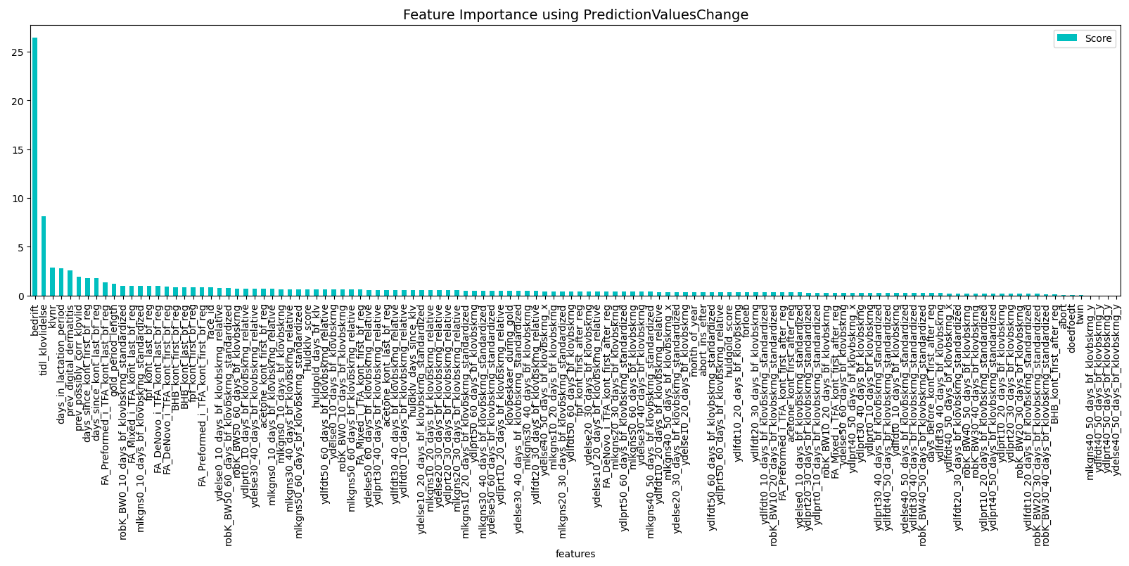


```
[ ]: X_test_subset = X_test
     fi = model.get_feature_importance(Pool(X_test_subset,␣
      ↪label=y_test,cat_features=cat_col_indices),
                                                                              ␣
      ↪type='PredictionValuesChange')
     feature_score = pd.DataFrame(list(zip(X_test_subset.dtypes.index, fi )),
                                     columns=['Feature','Score'])

     feature_score = feature_score.sort_values(by='Score', ascending=False,␣
      ↪inplace=False, kind='quicksort', na_position='last')

     plt.rcParams["figure.figsize"] = (20,5)
     ax = feature_score.plot('Feature', 'Score', kind='bar', color='c')
```

```
ax.set_title("Feature Importance using {}".format('PredictionValuesChange'),␣
    ↪fontsize = 14)
ax.set_xlabel("features")
plt.show()
```



1.1.8 Feature elimination

As seen below, lots of the model features are highly correlated. Among the 124 features, we train a model in a 20 steps, eliminating the least important features in each iteration until we are left with the 20 most important features.

[ ]:
```
# Pairwise feature correlation using pearson correlation coefficient
corr_matrix = X_train.corr().abs()
# Select upper triangle of correlation matrix
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))
# High correlation
high_correlation = [column for column in upper.columns if any(upper[column] > 0.
    ↪85)]
len(high_correlation)
```

```
The default value of numeric_only in DataFrame.corr is deprecated. In a future
version, it will default to False. Select only valid columns or specify the
value of numeric_only to silence this warning.
```

[ ]:
```
64
```

[ ]:
```
cat_col_indices_e = np.where(X_train.dtypes == object)[0]

model_elim = CatBoostClassifier(
    #class_weights=(1,10),
    n_estimators=500
)

train_pool = Pool(X_train, label=y_train,cat_features=cat_col_indices_e)
test_pool = Pool(X_test, label=y_test,cat_features=cat_col_indices_e)

summary_e = model_elim.select_features(
    train_pool,
    eval_set=test_pool,
    #features_for_select='0-186',# 124
    features_for_select='0-123',# 124
    num_features_to_select=20,
    steps=20,
    algorithm=EFeaturesSelectionAlgorithm.RecursiveByShapValues,
    shap_calc_type=EShapCalcType.Regular,
    train_final_model=True,
    verbose=False)
```

```
Learning rate set to 0.155339
Step #1 out of 20

bestTest = 0.518919663
bestIteration = 398

Shrink model to first 399 iterations.
Feature #43 eliminated
Feature #6 eliminated
Feature #45 eliminated
Feature #102 eliminated
Feature #19 eliminated
Feature #35 eliminated
Feature #29 eliminated
Feature #104 eliminated
Feature #27 eliminated
Feature #33 eliminated
Feature #44 eliminated
Step #2 out of 20

bestTest = 0.5184171518
bestIteration = 459

Shrink model to first 460 iterations.
Feature #109 eliminated
Feature #66 eliminated
Feature #40 eliminated
Feature #63 eliminated
Feature #56 eliminated
Feature #111 eliminated
Feature #5 eliminated
Feature #52 eliminated
Feature #39 eliminated
Feature #18 eliminated
Step #3 out of 20

bestTest = 0.5184196279
bestIteration = 480

Shrink model to first 481 iterations.
Feature #31 eliminated
Feature #57 eliminated
Feature #69 eliminated
Feature #113 eliminated
Feature #36 eliminated
Feature #13 eliminated
Feature #32 eliminated
Feature #24 eliminated
Feature #16 eliminated
Step #4 out of 20

bestTest = 0.5177692899
bestIteration = 452

Shrink model to first 453 iterations.
Feature #61 eliminated
Feature #50 eliminated
Feature #22 eliminated
Feature #25 eliminated
Feature #12 eliminated
Feature #60 eliminated
Feature #119 eliminated
Feature #53 eliminated
Step #5 out of 20

bestTest = 0.5167261365
bestIteration = 498

Shrink model to first 499 iterations.
Feature #107 eliminated
Feature #15 eliminated
Feature #59 eliminated
Feature #42 eliminated
Feature #37 eliminated
Feature #14 eliminated
Feature #9 eliminated
```

```
Step #6 out of 20

bestTest = 0.5166798666
bestIteration = 498

Shrink model to first 499 iterations.
Feature #3 eliminated
Feature #67 eliminated
Feature #38 eliminated
Feature #11 eliminated
Feature #54 eliminated
Feature #58 eliminated
Feature #121 eliminated
Step #7 out of 20

bestTest = 0.5166229211
bestIteration = 493

Shrink model to first 494 iterations.
Feature #51 eliminated
Feature #34 eliminated
Feature #74 eliminated
Feature #17 eliminated
Feature #48 eliminated
Feature #47 eliminated
Feature #103 eliminated
Step #8 out of 20

bestTest = 0.5166828622
bestIteration = 490

Shrink model to first 491 iterations.
Feature #68 eliminated
Feature #46 eliminated
Feature #7 eliminated
Feature #100 eliminated
Feature #95 eliminated
Step #9 out of 20

bestTest = 0.5165271416
bestIteration = 492

Shrink model to first 493 iterations.
Feature #41 eliminated
Feature #21 eliminated
Feature #91 eliminated
Feature #123 eliminated
Feature #23 eliminated
```

```
Step #10 out of 20

bestTest = 0.5166743485
bestIteration = 423

Shrink model to first 424 iterations.
Feature #92 eliminated
Feature #82 eliminated
Feature #117 eliminated
Feature #96 eliminated
Feature #77 eliminated
Step #11 out of 20

bestTest = 0.5163188916
bestIteration = 491

Shrink model to first 492 iterations.
Feature #10 eliminated
Feature #4 eliminated
Feature #65 eliminated
Feature #78 eliminated
Feature #79 eliminated
Step #12 out of 20

bestTest = 0.5156703716
bestIteration = 475

Shrink model to first 476 iterations.
Feature #85 eliminated
Feature #80 eliminated
Feature #88 eliminated
Feature #26 eliminated
Step #13 out of 20

bestTest = 0.5162463711
bestIteration = 498

Shrink model to first 499 iterations.
Feature #73 eliminated
Feature #55 eliminated
Feature #75 eliminated
Step #14 out of 20

bestTest = 0.5158058382
bestIteration = 499

Feature #106 eliminated
Feature #94 eliminated
Feature #89 eliminated
Step #15 out of 20

bestTest = 0.5159294808
bestIteration = 474

Shrink model to first 475 iterations.
Feature #118 eliminated
Feature #84 eliminated
Feature #62 eliminated
Step #16 out of 20

bestTest = 0.5158651282
bestIteration = 499

Feature #114 eliminated
Feature #30 eliminated
Feature #110 eliminated
Step #17 out of 20

bestTest = 0.514867389
bestIteration = 482

Shrink model to first 483 iterations.
Feature #90 eliminated
Feature #105 eliminated
Feature #72 eliminated
Step #18 out of 20

bestTest = 0.5150573024
bestIteration = 488

Shrink model to first 489 iterations.
Feature #120 eliminated
Feature #116 eliminated
Step #19 out of 20

bestTest = 0.5147930081
bestIteration = 499

Feature #122 eliminated
Feature #93 eliminated
Step #20 out of 20

bestTest = 0.5157691603
bestIteration = 499

Feature #28 eliminated
Feature #49 eliminated
Train final model

bestTest = 0.5160515145
bestIteration = 499
```

[ ]:
```
selected_features = summary_e['selected_features_names']
selected_features
```

[ ]:
```
['klvnr',
 'race_id',
 'ydelse0_10_days_bf_klovbskrng',
 'ydlprt0_10_days_bf_klovbskrng',
 'mlkgns0_10_days_bf_klovbskrng',
 'mlkgns0_10_days_bf_klovbskrng_standardized',
 'days_in_lactation_period',
 'robK_BW0_10_days_bf_klovbskrng',
 'robK_BW50_60_days_bf_klovbskrng',
 'robK_BW0_10_days_bf_klovbskrng_standardized',
 'robK_BW20_30_days_bf_klovbskrng_standardized',
 'robK_BW50_60_days_bf_klovbskrng_standardized',
 'bedrift',
 'tidl_klovlidelse',
 'prev_possibly_corr_klovlid',
 'prev_digital_dermatitis',
 'klovbeskaer_during_gold',
 'days_since_kont_first_bf_reg',
 'FA_DeNovo_i_TFA_kont_last_bf_reg',
 'days_since_kont_last_bf_reg']
```

[ ]:
```
X_train_subset = X_train[selected_features]
X_test_subset = X_test[selected_features]
cat_col_indices = np.where(X_train_subset.dtypes == object)[0]

model_selected = CatBoostClassifier(
    #class_weights=(1,10),
    n_estimators=20000
)

model_selected.fit(
    X_train_subset,
    y_train,
    eval_set=(X_test_subset, y_test),
    verbose=False,
    cat_features=cat_col_indices,
    early_stopping_rounds=200
)
```

[ ]:

```
<catboost.core.CatBoostClassifier at 0x1e8e66dbc10>
```

[ ]:

```python
y_pred_prob = model_selected.predict_proba(X_test_subset, )[:,1]
y_pred = y_pred_prob >= 0.5
```

[ ]:

```python
pp(classification_report(y_true=y_test.values,y_pred=y_pred))
```

```
('              precision    recall  f1-score   support\n'
 '\n'
 '       False       0.77      0.88      0.82     40834\n'
 '        True       0.67      0.47      0.55     20507\n'
 '\n'
 '    accuracy                           0.74     61341\n'
 '   macro avg       0.72      0.68      0.69     61341\n'
 'weighted avg       0.73      0.74      0.73     61341\n')
```

[ ]:

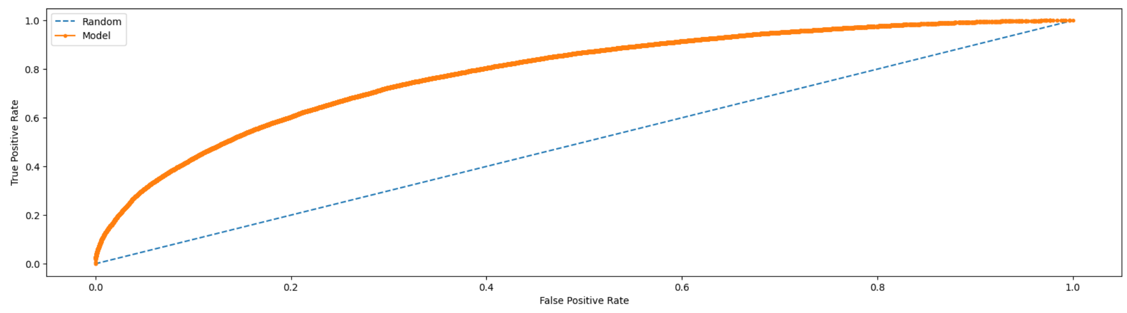
```python
# Area under ROC curve
roc_auc_score(y_score=y_pred_prob, y_true=y_test)
```

[ ]:

```
0.7853673889122623
```

[ ]:

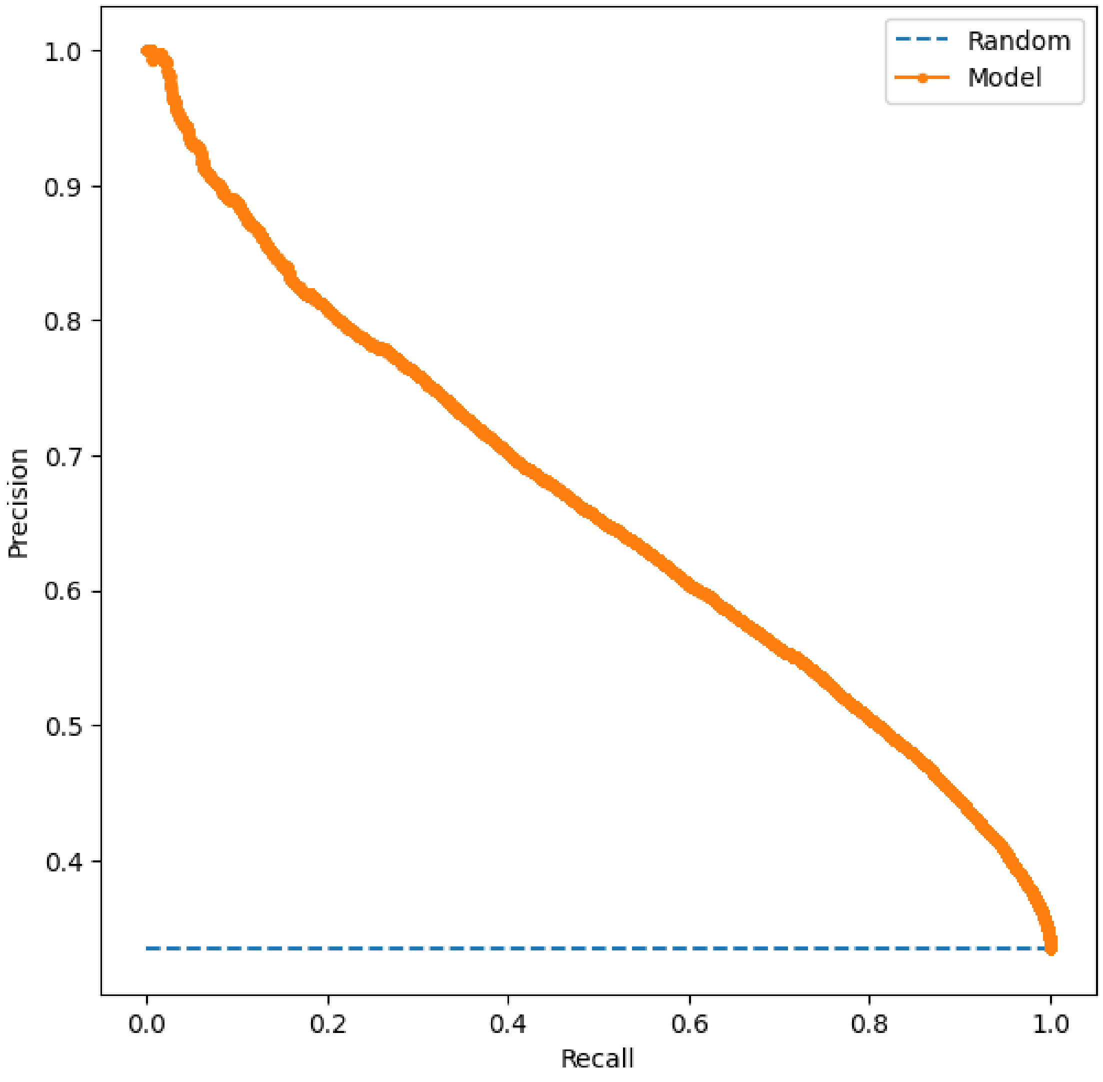
```python
# Plot ROC curve
random_probs = [0 for _ in range(len(y_test))]
model_fpr, model_tpr, _ = roc_curve(y_test, y_pred_prob)
random_fpr, random_tpr, _ = roc_curve(y_test, random_probs)
plt.plot(random_fpr, random_tpr, linestyle='--', label='Random')
plt.plot(model_fpr, model_tpr, marker='.', label='Model')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.rcParams["figure.figsize"] = (7,7)
plt.show()
```



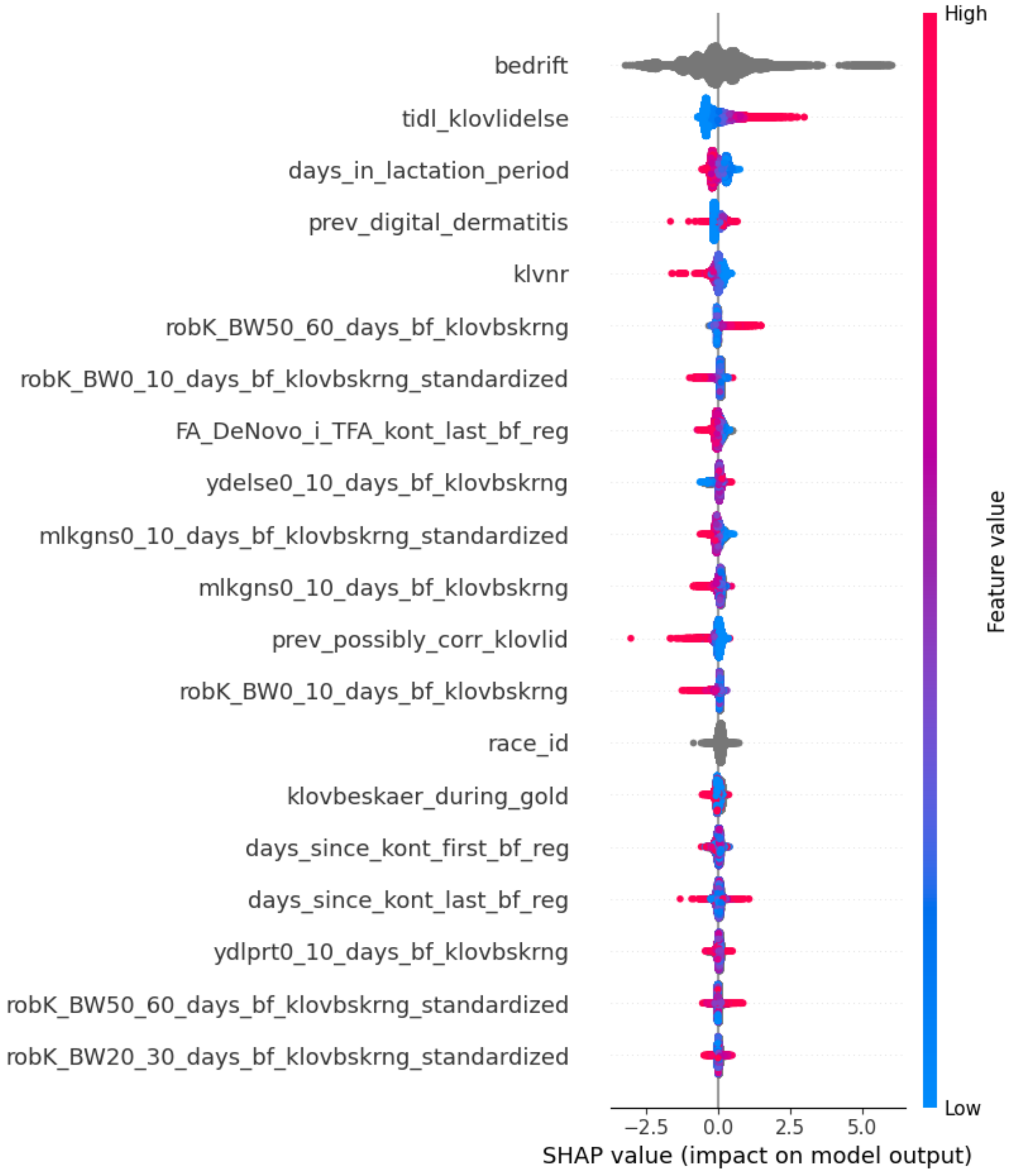
```
[ ]: # Precision-recall curve
     precision, recall, thresholds = precision_recall_curve(y_test, y_pred_prob)
     # Area under precision-recall curve
     auc(recall, precision)
```

```
[ ]: 0.6595349089242456
```

```
[ ]: # Plot precision-recall curve
     random = len(y_test[y_test==1]) / len(y_test)
     plt.plot([0, 1], [random, random], linestyle='--', label='Random')
     plt.plot(recall, precision, marker='.', label='Model')
     plt.xlabel('Recall')
     plt.ylabel('Precision')
     plt.legend()
     plt.rcParams["figure.figsize"] = (7,7)
     plt.show()
```



```
[ ]: shap_values = model_selected.get_feature_importance(Pool(X_test_subset,␣
      ↪label=y_test,cat_features=cat_col_indices),                                 ␣
      ↪type="ShapValues")
     shap.summary_plot(shap_values[:,:-1], X_test_subset, max_display=1000)
```

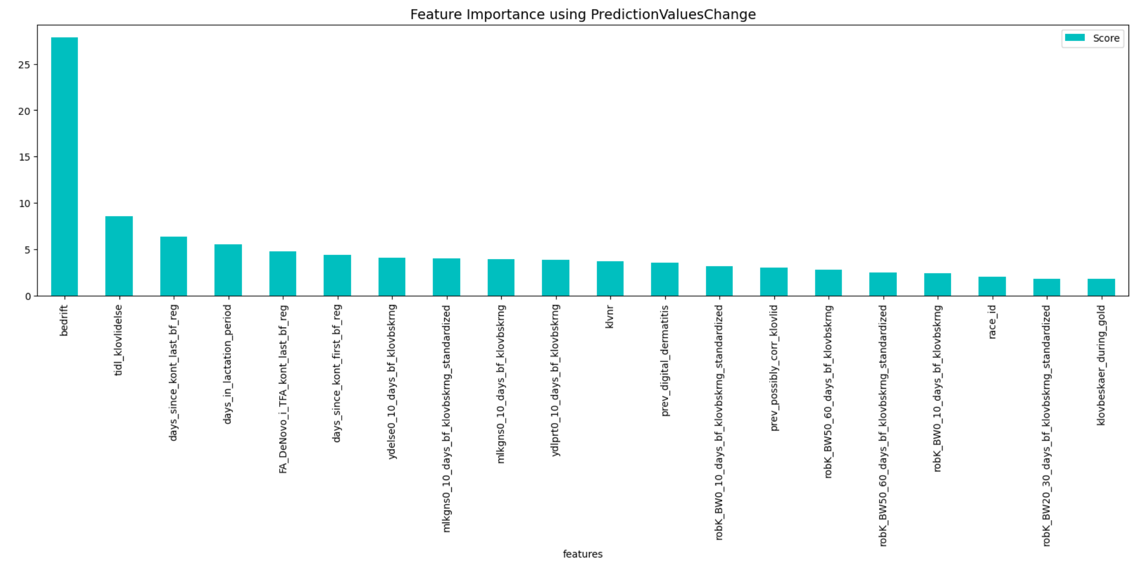


```
[ ]: fi = model_selected.get_feature_importance(Pool(X_test_subset,␣
      ↪label=y_test,cat_features=cat_col_indices),                                                       ␣

      ↪type='PredictionValuesChange')
     feature_score = pd.DataFrame(list(zip(X_test_subset.dtypes.index, fi )),
                                     columns=['Feature','Score'])

     feature_score = feature_score.sort_values(by='Score', ascending=False,␣
      ↪inplace=False, kind='quicksort', na_position='last')

     plt.rcParams["figure.figsize"] = (20,5)
     ax = feature_score.plot('Feature', 'Score', kind='bar', color='c')
     ax.set_title("Feature Importance using {}".format('PredictionValuesChange'),␣
      ↪fontsize = 14)
     ax.set_xlabel("features")
     plt.show()
```



Hyperparameter tuning - Randomized Search Cross Validation Attempt to find better subset of hyperparameters (learning rate and max_depth values) to improve model accuracy

```
[ ]: from sklearn.model_selection import RandomizedSearchCV
     from scipy.stats import randint

     cbc = CatBoostClassifier(n_estimators=20000)

     # Creating the hyperparameter grid
     param_dist = { "learning_rate": np.linspace(0,0.2,5),
                    "max_depth": randint(3, 10)}
```

```
#Instantiate RandomSearchCV object
rscv = RandomizedSearchCV(cbc , param_dist, scoring='accuracy', cv =5)

#Fit the model
rscv.fit(
    X_train_subset,
    y_train.astype(str),
    eval_set=(X_test_subset, y_test.astype(str)),
    verbose=1,
    cat_features=cat_col_indices,
    early_stopping_rounds=200
)

# Print the tuned parameters and score
print(rscv.best_estimator_)
print(rscv.best_params_)
print(rscv.best_score_)
# optimalt params: {'learning_rate': 0.05, 'max_depth': 7}
# accuracy_score: 0.7413801546735949
```

[ ]:
```
y_pred_prob_rscv = rscv.best_estimator_.predict_proba(X_test_subset, )[:,1]
y_pred_rscv = y_pred_prob_rscv >= 0.5
```

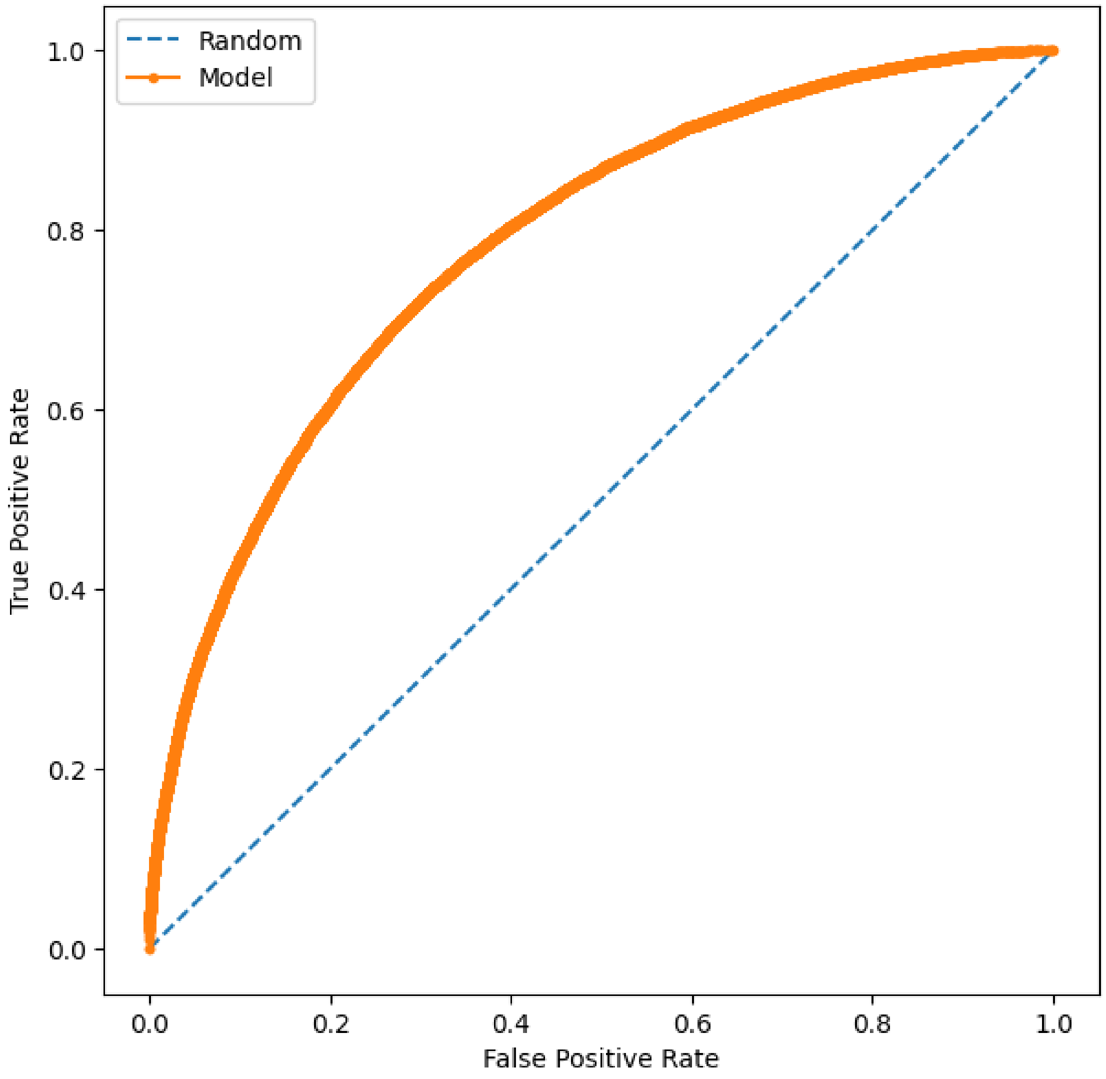
[ ]:
```
# Area under ROC curve
roc_auc_score(y_score=y_pred_prob_rscv, y_true=y_test)
```

[ ]:
```
0.7850317753944702
```

[ ]:
```
pp(classification_report(y_true=y_test.values,y_pred=y_pred_rscv))
```

```
('              precision    recall  f1-score   support\n'
 '\n'
 '       False       0.77      0.88      0.82     40834\n'
 '        True       0.67      0.47      0.55     20507\n'
 '\n'
 '    accuracy                           0.74     61341\n'
 '   macro avg       0.72      0.68      0.69     61341\n'
 'weighted avg       0.73      0.74      0.73     61341\n')
```

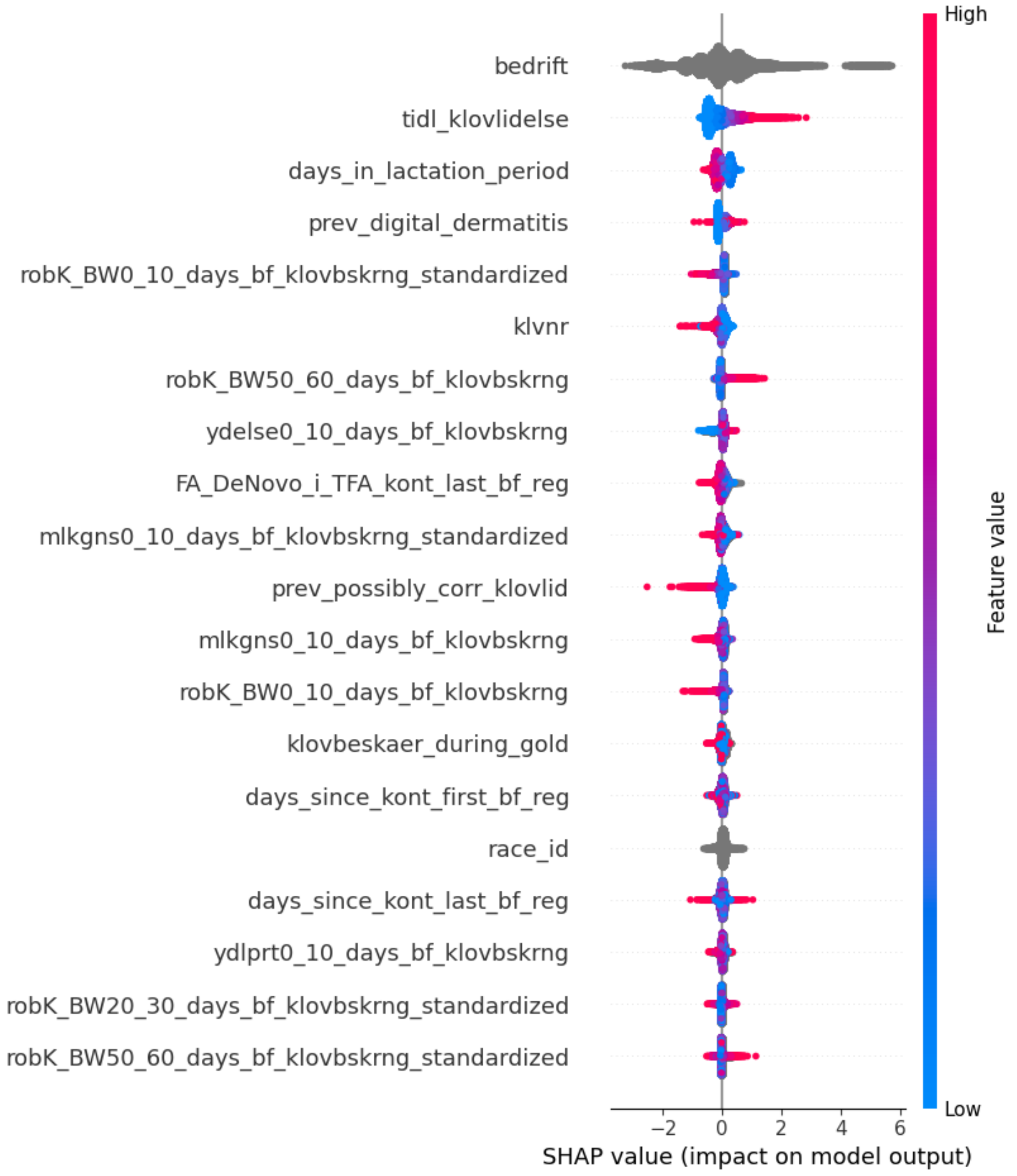
[ ]:
```
# Plot ROC curve
random_probs = [0 for _ in range(len(y_test))]
model_fpr, model_tpr, _ = roc_curve(y_test, y_pred_prob_rscv)
random_fpr, random_tpr, _ = roc_curve(y_test, random_probs)
plt.plot(random_fpr, random_tpr, linestyle='--', label='Random')
plt.plot(model_fpr, model_tpr, marker='.', label='Model')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.rcParams["figure.figsize"] = (7,7)
plt.show()
```



[ ]:
```
# Precision-recall curve
precision, recall, thresholds = precision_recall_curve(y_test, y_pred_prob)
# Area under precision-recall curve
auc(recall, precision)
```

[ ]:
```
0.6595349089242456
```

```
[ ]: shap_values_rscv = rscv.best_estimator_.
      ↪get_feature_importance(Pool(X_test_subset,␣
      ↪label=y_test,cat_features=cat_col_indices),                                    ␣
      ↪type="ShapValues")
     shap.summary_plot(shap_values_rscv[:,:-1], X_test_subset, max_display=1000)
```



1.1.9 Classification inspection and performance improvement by modelling besaetninger/bedrifter with presumably similar management

Attempt find besaetninger/bedrifter with similar management to improve model performance

```
[ ]: def measures(df):
         no_obs = int(len(df))
         num_klovlid = int(len(df[df['target']==True]))

         if len(df['target'].unique())==1:
             roc_auc = None
             pr_auc = None
             precision = None
             recall = None
             fscore = None
         else:
             roc_auc = roc_auc_score(y_score=df['pred_prob'], y_true=df['target'])
             precision, recall, thresholds = precision_recall_curve(df['target'],␣
      ↪df['pred_prob'])
             pr_auc = round(auc(recall, precision),4)
             scores = precision_recall_fscore_support(y_true=df['target'].
      ↪values,y_pred=df['pred'].values)
             precision =scores[0][1]
             recall = scores[1][1]
             fscore = scores[2][1]

         return pd.Series((no_obs, num_klovlid, pr_auc, roc_auc, precision, recall,␣
      ↪fscore), index=['no_obs', 'num_klovlid', 'PR_AUC', 'ROC_AUC', 'precision',␣
      ↪'recall', 'f1'])
```

```
[ ]: test_set = X_test.merge(y_test, how='inner', left_index=True, right_index=True)
     test_set['pred']=y_pred_rscv
     test_set['pred_prob']=y_pred_prob_rscv
```

Performance across 'besaetninger/bedrifter'

```
[ ]: with warnings.catch_warnings(record=True):
         bedrifter_compare = test_set.groupby('bedrift').apply(measures).
      ↪reset_index(drop=True)
     print(bedrifter_compare.iloc[:,0:5].sort_values(by='PR_AUC',ascending=False).
      ↪head(40).to_markdown(index=False))
```

```
|   no_obs |   num_klovlid |   PR_AUC |   ROC_AUC |   precision |
|---------:|--------------:|---------:|----------:|------------:|
|       68 |            67 |   0.9991 |  0.940299 |    0.985294 |
|       43 |            41 |   0.9905 |  0.817073 |    0.953488 |
|       27 |            26 |   0.9865 |  0.692308 |    0.962963 |
|        8 |               6 |   0.9742 |  0.916667 |    1        |
|      282 |             247 |   0.9422 |  0.726894 |    0.875887 |
|      157 |             114 |   0.8997 |  0.791922 |    0.726115 |
|      214 |             185 |   0.8982 |  0.606337 |    0.864486 |
|      387 |             302 |   0.898  |  0.710986 |    0.797872 |
|       58 |              49 |   0.8838 |  0.512472 |    0.844828 |
|       73 |              57 |   0.8831 |  0.692982 |    0.8      |
|       66 |              48 |   0.878  |  0.695602 |    0.727273 |
|      136 |              87 |   0.8445 |  0.754164 |    0.663934 |
|      222 |             171 |   0.8415 |  0.612888 |    0.769231 |
|       79 |              53 |   0.8256 |  0.729318 |    0.684211 |
|      316 |             242 |   0.8172 |  0.591803 |    0.765823 |
|      110 |              75 |   0.8102 |  0.653714 |    0.790698 |
|      351 |             256 |   0.7994 |  0.609581 |    0.738235 |
|      609 |             424 |   0.7914 |  0.681221 |    0.73913  |
|      357 |             212 |   0.7851 |  0.748178 |    0.759615 |
|      194 |             104 |   0.7849 |  0.777137 |    0.678832 |
|      158 |             105 |   0.7827 |  0.66469  |    0.668919 |
|      296 |             186 |   0.7801 |  0.678055 |    0.65019  |
|      341 |             236 |   0.7789 |  0.651453 |    0.722397 |
|       36 |              22 |   0.7786 |  0.633117 |    0.642857 |
|       89 |              47 |   0.777  |  0.74924  |    0.740741 |
|      410 |             221 |   0.7755 |  0.748163 |    0.70614  |
|      892 |             581 |   0.7738 |  0.651195 |    0.697987 |
|       84 |              39 |   0.7704 |  0.811966 |    0.8      |
|        8 |               3 |   0.7639 |  0.866667 |    0        |
|      106 |              56 |   0.7606 |  0.75     |    0.677419 |
|      384 |             212 |   0.7531 |  0.688405 |    0.637066 |
|       26 |              12 |   0.7456 |  0.744048 |    0.8      |
|      101 |              64 |   0.7397 |  0.694257 |    0.633663 |
|      280 |             176 |   0.7358 |  0.63396  |    0.655319 |
|      403 |             221 |   0.7344 |  0.68833  |    0.638554 |
|      158 |              79 |   0.7333 |  0.726967 |    0.710526 |
|      142 |              92 |   0.7322 |  0.639783 |    0.656489 |
|      139 |              70 |   0.7284 |  0.70559  |    0.709091 |
|      265 |             133 |   0.728  |  0.71309  |    0.598765 |
|      529 |             323 |   0.7208 |  0.619571 |    0.632967 |
```

As can be seen, performance varies greatly across 'bedrifter'. This is most likely due to the management aspect which becomes present in the data.

```
[ ]: relative_features = ['ydelse','robK_BW', 'ydlprt', 'mlkgns']
     missing_values_cols = [name + '50_60_days_bf_klovbskrng' for name in␣
      ↪relative_features]
     test_set['missing'] = test_set[missing_values_cols].isnull().any(axis=1).values
```

```
[ ]: missing_compare = test_set.groupby('missing').apply(measures).reset_index()
     pp(missing_compare.sort_values(by='PR_AUC',ascending=False))
```

```
   missing   no_obs  num_klovlid  PR_AUC   ROC_AUC  precision    recall  \
1     True  40613.0      13656.0  0.6579  0.783044   0.669628  0.458187
0    False  20728.0       6851.0  0.6494  0.779585   0.664336  0.443731

         f1
1  0.544087
0  0.532073
```

Using cross validation, the best performing 'bedrifter' are found. Similar performance is here used as an imperfect proxy for similar management.

```
[ ]: kf = KFold(n_splits=4, random_state=123, shuffle=True)
     X = df_full_dataset.drop(["target"], axis=1)
     y = df_full_dataset["target"]
     fold = 1
     folds_measures = []
     for train_index, test_index in kf.split(X):
         # print("TRAIN:", train_index, "TEST:", test_index)
         X_train, X_test = X.iloc[train_index], X.iloc[test_index]
         y_train, y_test = y[train_index], y[test_index]

         cat_col_indices = np.where(X_train.dtypes == object)[0]
         model_cv = CatBoostClassifier(
             #class_weights=(1,10),
             n_estimators=500
         )

         model_cv.fit(
             X_train,
             y_train,
             eval_set=(X_test, y_test),
             verbose=False,
             cat_features=cat_col_indices,
             early_stopping_rounds=100
         )

         y_pred_prob_cv = model_cv.predict_proba(X_test, )[:,1]
         y_pred_cv = y_pred_prob_cv >= 0.5

         test_set = X_test.merge(y_test, how='inner', left_index=True,␣
     ↪right_index=True)
         test_set['pred']=y_pred_cv
         test_set['pred_prob']=y_pred_prob_cv

         with warnings.catch_warnings(record=True):
             bedrifter_compare = test_set.groupby('bedrift').apply(measures).
     ↪reset_index().sort_values(by='PR_AUC',ascending=False)
         bedrifter_compare['fold'] = fold
```

```
    folds_measures.append(bedrifter_compare)
    fold += 1
```

[ ]:
```
all_folds = pd.concat(folds_measures, axis=0)
compare_bedrifter = all_folds.groupby('bedrift').mean().reset_index().
 ↪sort_values(by='PR_AUC',ascending=False)
print(compare_bedrifter.iloc[:40,1:7].to_markdown(index=False))
```

```
|    no_obs |    num_klovlid |    PR_AUC |    ROC_AUC |    precision |    recall |
|----------:|---------------:|----------:|-----------:|-------------:|----------:|
|   1.5     |       0.25     | 1         |  1         |     0        | 0         |
|  41.75    |       0.25     | 1         |  1         |     0        | 0         |
|   3.33333 |       0.333333 | 1         |  1         |     0        | 0         |
| 275.5     |     275        | 0.9991    |  0.797167  |     0.996315 | 1         |
|  25.5     |      25.25     | 0.9856    |  0.666667  |     0.964286 | 1         |
| 269       |     234.25     | 0.951375  |  0.772851  |     0.87242  | 0.996816  |
|  54.75    |      53.25     | 0.9504    |  0.547166  |     0.964336 | 1         |
| 179       |     140.75     | 0.9128    |  0.752652  |     0.794239 | 0.985722  |
| 196.75    |     169.5      | 0.906175  |  0.620561  |     0.860855 | 0.998656  |
|   9.25    |       1        | 0.902767  |  0.944444  |     0        | 0         |
| 399.75    |     306.5      | 0.882025  |  0.696819  |     0.773511 | 0.984943  |
|  45.5     |      38.75     | 0.880525  |  0.647935  |     0.852978 | 1         |
|  68.75    |      53.75     | 0.866975  |  0.63339   |     0.780136 | 1         |
|  59.75    |      46        | 0.85825   |  0.636976  |     0.765199 | 0.992647  |
|  96       |      69.25     | 0.8516    |  0.695733  |     0.736849 | 0.941639  |
|  63.75    |      47.5      | 0.837675  |  0.668345  |     0.760992 | 0.990625  |
| 330.5     |     254.75     | 0.83505   |  0.615861  |     0.769963 | 0.995165  |
| 232.5     |     178.5      | 0.820675  |  0.600038  |     0.766871 | 0.989675  |
| 323.5     |     233.75     | 0.798925  |  0.619706  |     0.727088 | 0.958067  |
| 660.75    |     458.25     | 0.795825  |  0.675119  |     0.729402 | 0.947229  |
| 135.5     |      89.5      | 0.7942    |  0.679038  |     0.690497 | 0.907086  |
|   6       |       3.5      | 0.7939    |  0.8       |     0.821429 | 0.445833  |
| 338       |     233.75     | 0.780125  |  0.65165   |     0.715714 | 0.951716  |
| 303       |     197.5      | 0.775725  |  0.670292  |     0.687289 | 0.944343  |
| 153.5     |      98.5      | 0.771575  |  0.669585  |     0.658642 | 0.935778  |
|  78.25    |      48.25     | 0.771475  |  0.671215  |     0.700889 | 0.746624  |
|  40.5     |      25.75     | 0.767625  |  0.665663  |     0.683329 | 0.816504  |
| 884.75    |     561        | 0.765925  |  0.660451  |     0.678781 | 0.89124   |
|  85.75    |      61        | 0.76345   |  0.577534  |     0.718325 | 0.946908  |
| 351.75    |     198        | 0.757     |  0.744972  |     0.703733 | 0.778854  |
| 132.75    |      85.25     | 0.75175   |  0.645777  |     0.659843 | 0.931785  |
|  84       |      41        | 0.749125  |  0.750068  |     0.736538 | 0.605952  |
| 194.25    |     113        | 0.748925  |  0.722199  |     0.68227  | 0.874508  |
|  51.75    |      28        | 0.74355   |  0.693543  |     0.659153 | 0.750576  |
| 108.25    |      61.5      | 0.74025   |  0.678477  |     0.691588 | 0.64196   |
|  73.75    |      32.25     | 0.73955   |  0.777751  |     0.794444 | 0.31141   |
| 419.25    |     224.5      | 0.73315   |  0.702068  |     0.652883 | 0.714588  |
| 537.25      |     336.5       | 0.72835  |  0.639359 |     0.666577 | 0.894934 |
| 108.5       |      57         | 0.72755  |  0.71085  |     0.652124 | 0.706322 |
| 280         |     145.75      | 0.724725 |  0.6942   |     0.641203 | 0.688049 |
```

[ ]:
```
best_bedrifter = compare_bedrifter[compare_bedrifter['ROC_AUC']>=0.
  ↪7]['bedrift'].values
print(len(best_bedrifter))
```

```
49
```

[ ]:
```
df_best_bedrifter = df_full_dataset[df_full_dataset['bedrift'].
  ↪isin(best_bedrifter)]
X_new = df_best_bedrifter.drop(["target"], axis=1)
y_new = df_best_bedrifter["target"]

X_train, X_test, y_train, y_test = train_test_split(
    X_new, y_new, test_size=0.2, random_state=123)

print(f'Length of test dataset: {len(y_test)}')
print(f'Percentage of hoof disorder cases: {round(len(y_test[y_test==1]) /␣
  ↪len(y_test), 2)}')
```

```
Length of test dataset: 9504
Percentage of hoof disorder cases: 0.34
```

Final model with the 20 most important features, and the 49 'most similar' besaetninger/bedrifter

[ ]:
```
X_train_subset = X_train[selected_features]
X_test_subset = X_test[selected_features]
cat_col_indices = np.where(X_train_subset.dtypes == object)[0]

model = CatBoostClassifier(
    #class_weights=(1,10),
    n_estimators=20000
)

model.fit(
    X_train_subset,
    y_train,
    eval_set=(X_test_subset, y_test),
    verbose=False,
    cat_features=cat_col_indices,
    early_stopping_rounds=200
)
```

[ ]:
```
<catboost.core.CatBoostClassifier at 0x7fb01ed62c40>
```

```
[ ]: y_pred_prob = model.predict_proba(X_test_subset, )[:,1]
     y_pred = y_pred_prob >= 0.5
```

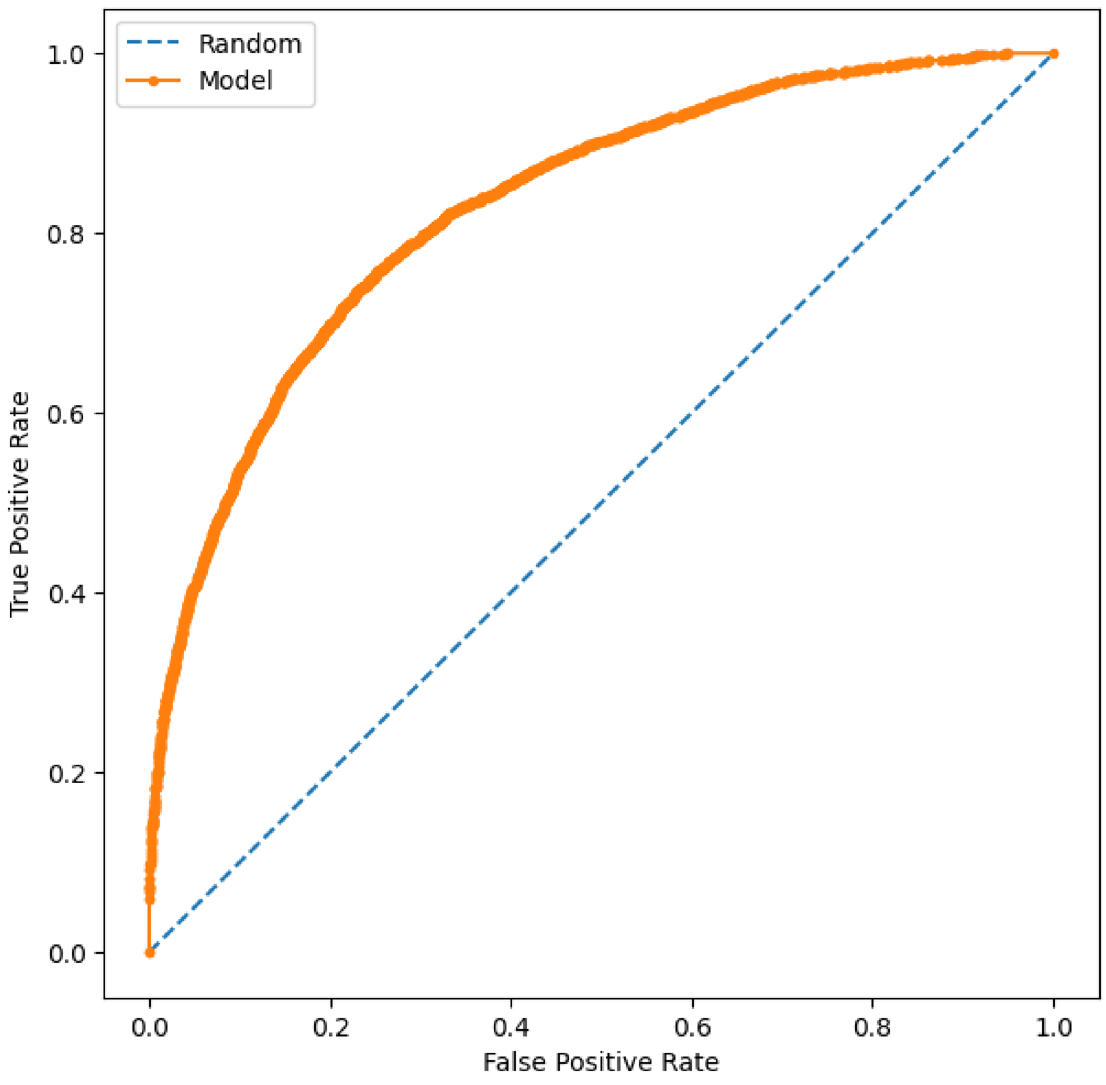
```
[ ]: pp(classification_report(y_true=y_test.values,y_pred=y_pred))
```

```
('              precision    recall  f1-score   support\n'
 '\n'
 '       False       0.80      0.89      0.84      6312\n'
 '        True       0.72      0.55      0.62      3192\n'
 '\n'
 '    accuracy                           0.78      9504\n'
 '   macro avg       0.76      0.72      0.73      9504\n'
 'weighted avg       0.77      0.78      0.77      9504\n')
```

```
[ ]: # Area under ROC curve
     roc_auc_score(y_score=y_pred_prob, y_true=y_test)
```

```
[ ]: 0.8303457769105909
```

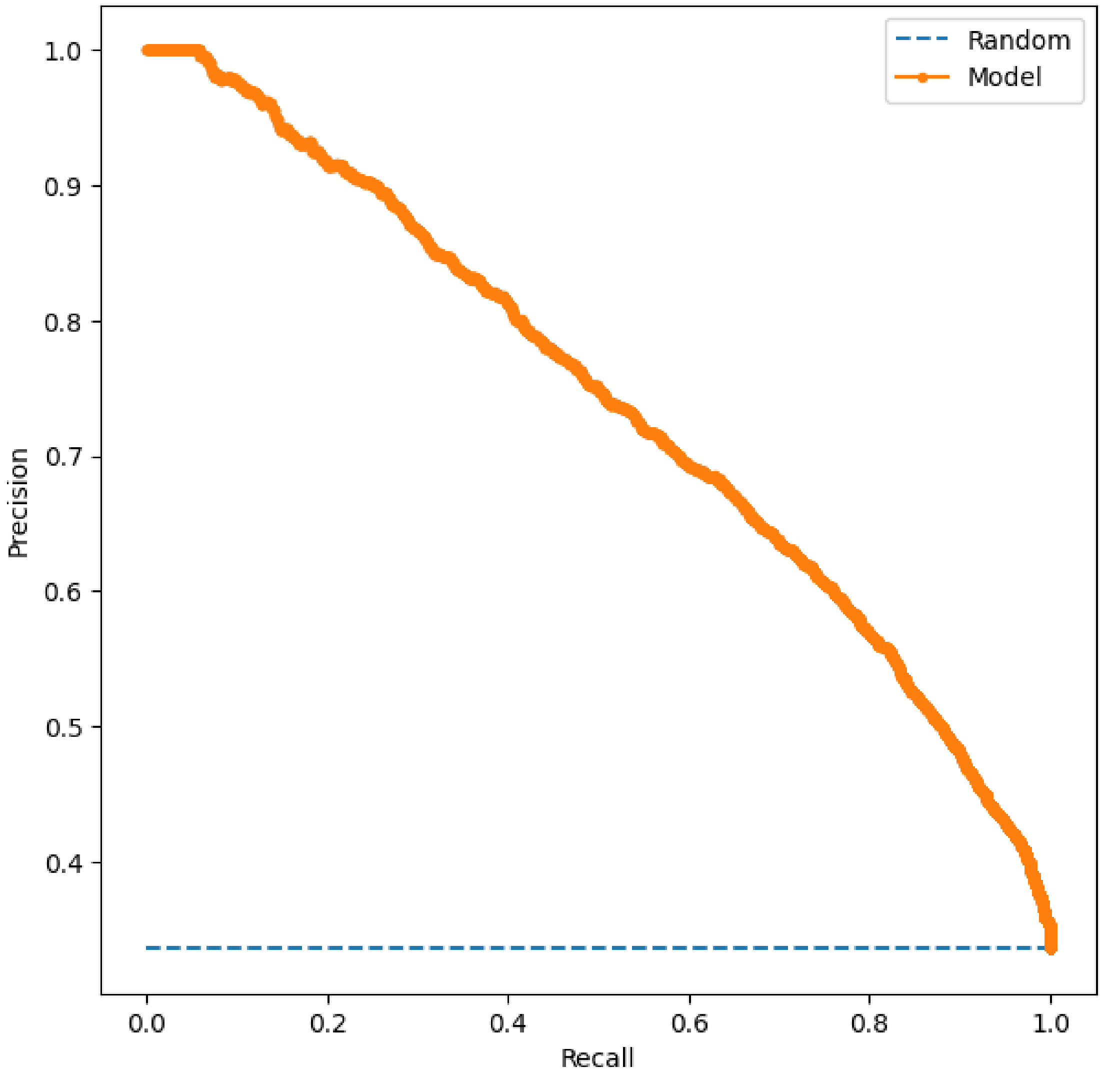
```
[ ]: # Plot ROC curve
     random_probs = [0 for _ in range(len(y_test))]
     model_fpr, model_tpr, _ = roc_curve(y_test, y_pred_prob)
     random_fpr, random_tpr, _ = roc_curve(y_test, random_probs)
     plt.plot(random_fpr, random_tpr, linestyle='--', label='Random')
     plt.plot(model_fpr, model_tpr, marker='.', label='Model')
     plt.xlabel('False Positive Rate')
     plt.ylabel('True Positive Rate')
     plt.legend()
     plt.rcParams["figure.figsize"] = (7,7)
     plt.show()
```



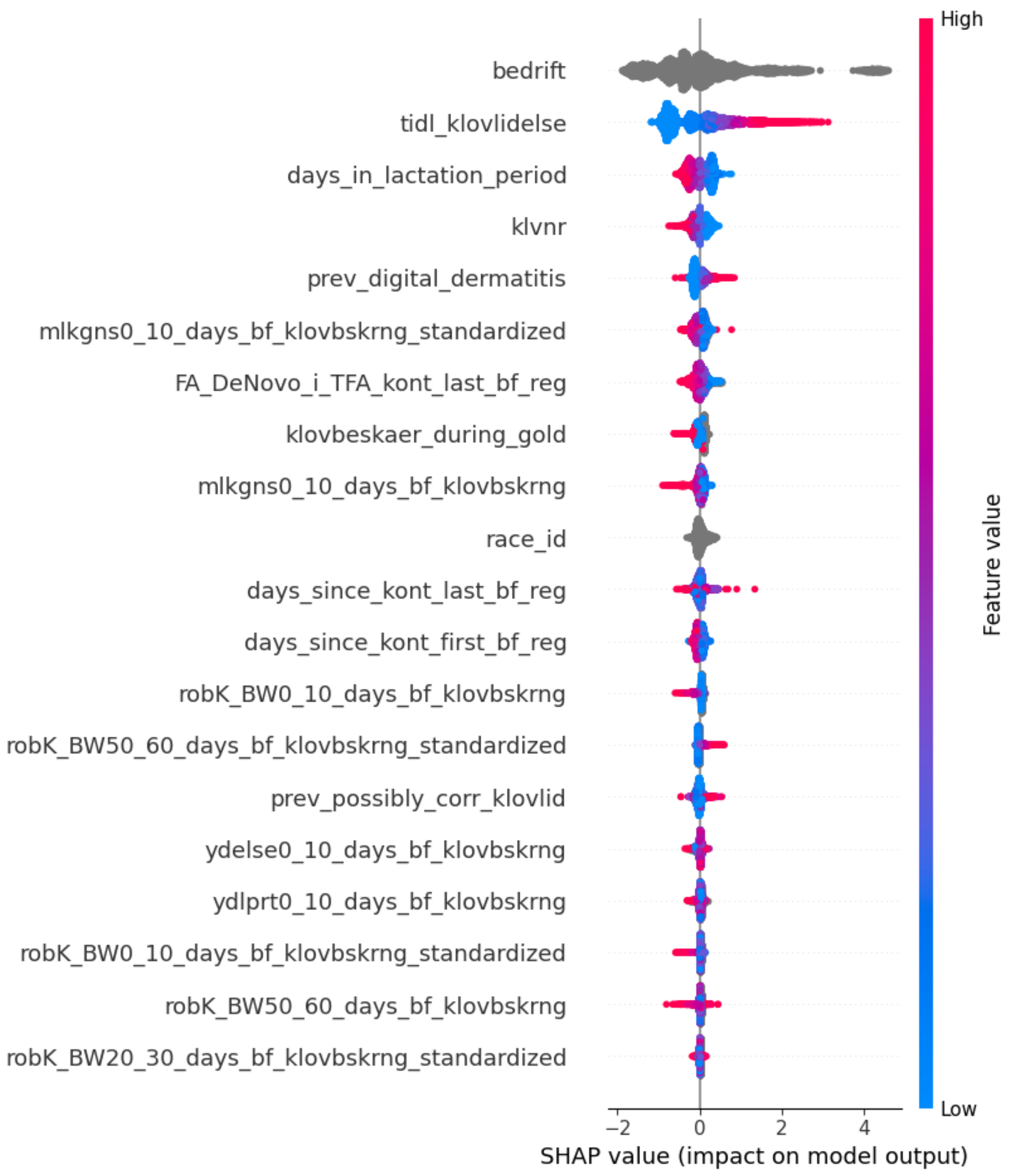
```
[ ]: # Precision-recall curve
     precision, recall, thresholds = precision_recall_curve(y_test, y_pred_prob)
     # Area under precision-recall curve
     auc(recall, precision)
```

```
[ ]: 0.7395635736840505
```

```
[ ]: # Plot precision-recall curve
     random = len(y_test[y_test==1]) / len(y_test)
     plt.plot([0, 1], [random, random], linestyle='--', label='Random')
     plt.plot(recall, precision, marker='.', label='Model')
     plt.xlabel('Recall')
     plt.ylabel('Precision')
     plt.legend()
     plt.rcParams["figure.figsize"] = (7,7)
     plt.show()
```



```
[ ]: shap_values_final = model.get_feature_importance(Pool(X_test_subset,␣
      ↪label=y_test,cat_features=cat_col_indices),
                                                                                 ␣
      ↪type="ShapValues")
     shap.summary_plot(shap_values_final[:,:-1], X_test_subset, max_display=1000)
```



bedrift
tidl_klovlidelse
days_in_lactation_period
klvnr
prev_digital_dermatitis
mlkgns0_10_days_bf_klovbskrng_standardized
FA_DeNovo_i_TFA_kont_last_bf_reg
klovbeskaer_during_gold
mlkgns0_10_days_bf_klovbskrng
race_id
days_since_kont_last_bf_reg
days_since_kont_first_bf_reg
robK_BW0_10_days_bf_klovbskrng
robK_BW50_60_days_bf_klovbskrng_standardized
prev_possibly_corr_klovlid
ydelse0_10_days_bf_klovbskrng
ydlprt0_10_days_bf_klovbskrng
robK_BW0_10_days_bf_klovbskrng_standardized
robK_BW50_60_days_bf_klovbskrng
robK_BW20_30_days_bf_klovbskrng_standardized
−2
0
2
4
SHAP value (impact on model output)
High
Low
Feature value

```
[ ]:
```