Forudsigelse af ketose hos malkekøer	Ansvarlig	THA
	Oprettet	15-12-2021
Projekt: 5517, Datadrevet management i mælkeproduktionen	Side	1 af 1

STØTTET AF
mælkeafgiftsfonden

Forudsigelse af ketose på basis af data fra kvægdatabasen

Ketose er en stofskiftesygdom, der opstår, hvis en ko er i negativ energibalance og har en høj mobilisering. Typisk opstår problemet først i laktationen. Symptomerne for ketose er ofte kraftigt vægttab, nedsat foderoptagelse, ydelsestab og acetone lugt i udåndingsluft.

Vedlagte notat indeholder en detaljeret beskrivelse af de eksperimenter der er gennemført for at træne en maskinlærings model til at forudsige ketose hos køer i den første del af laktationen.

DAT-1937 Prediction of Ketosis in Cows

December 14, 2021

This notebook details an experiment on training and validating a machine learning model for predicting ketosis in cows.
Contact information: SEGES-datascience@seges.dk

1 Setup

1.1 Raw data

The following data files are used across all 'bedrifter':

- hendelserprdag{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.

1.2 Target

A binary target specifying whether the cow gets ketosis within the first 24 days after calving.
Thus, for each cow and calving, a single data point is created.

1.3 Features

1.3.1 From hendelserprdag{bedrift}.csv

- Ydelse_total, robK_BW, MILKPROTEIN, MILKFAT, and fpf (MILKFAT/MILKPROTEIN) for the first 10 observations after calving.
- The relative change of the above mentioned variables after calving compared to the first observation after calving.
- The average value of Ydelse_total, robK_BW, MILKPROTEIN, MILKFAT, and fpf for five 30-day periods before calving (thus, data from 150 days before calving is used).
- The relative change of of the above mentioned variables up until calving compared to the [150 days - 120 days before calving] average.
- 'gold'-period length.
- 'bedrift'.
- race_id.
- klvnr.

- The average value of MILKSPEEDMAX, MILKTIMEKO, LFDEADMILKTIME, MILKTEMPERATURE, RFDEADMILKTIME, LRDEADMILKTIME, and RRDEADMILKTIME for the first 10 observations after calving.
- The difference between the weight before the 'gold' period and after calving.
- Month of the year (seasonality).

1.3.2 From syg{bedrift}.csv

- The number of times the cow has had ketosis previously.
- If the cow currently has or has had 'efterbyrd', 'kælvningsfeber', 'børbetændelse' since the last calving.

1.3.3 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 the last 'laktation', and for the first inspection after calving if the inspection is close to the calving date.

1.3.4 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 when calving.
- The time in days from when both 'huld' scores were given until the calving date.

1.4 Limitations

At least a single data point from the previous 'laktation' needs to exist for each cow to be considered. This also means that the first calving for a cow is not considered in the model.

1.5 Models

The following model is used:

- Gradient tree boosting model - catboost

1.6 Performance Metrics

The model is evaluated based on the following criteria:

- Precision-recall AUC
- Precision
- Recall
- F1
- ROC AUC

2 Experiments

- First, a model is trained on data across all 'bedrifter' and validated on a random subset of the data (a 75/25 train/validation split of the data). Early stopping together with the validation

set is used to find the optimal number of iterations (trees) in the model. Class weights are used, where the minority class (target=1, meaning that the cow gets ketosis) is giving a factor 10 weight. Otherwise, the standard parameters in catboost are used.

- Next, the performance is evaluated. Performance across 'bedrifter' is examined together with performance of cows with 150 days of daily history before calving compared with cows with less than that.
- Using cross validation, the best performing 'bedrifter' are found. Similar performance is here used as an imperfect proxy for similar management.
- The best performing 'bedrifter' is chosen, and the model is trained only on these 'bedrifter' and validated on a random subset of the data (a 80/20 train/validation split of the data).
- Next, the performance is evaluated again.
- The features are then examined to determine the most important features and their effect on the target.
- The least important features are then removed and a new model is trained. This is done in 20 steps, where a small subset of features is removed for each run.
- The performance is evaluated again.
- Finally, the trade-off in the precision and recall are examined to find suitable cutoffs depending on priorities.

3 Conclusions

- The data shows significant differences between 'bedrifter'. This suggests that the 'management' in the individual 'bedrifter' has a large impact on the results. Data from different 'bedrifter' may therefore not be comparable, and a model including data from all 'bedrifter' may not be suitable.
- This is also related to the fact that the target value is based on treatment registrations and not illness registrations. Thus, some cows may be ill but does not show strong enough symptoms of illness to receive treatment. Furthermore, some cows may receive the treatment by the farmer himself without having a veterinarian registration. This does not show up in the data and has a large effect on the performance of the model. This is related to the specific 'management' of the 'bedrift', and it may explain the large differences in the data and performance of the model across 'bedrifter'.
- Training on only 'similar' (the best performing) 'bedrifter' greatly improves the performance.
- Performance is also significantly better for cows with at least 150 days of daily history before calving compared to cows with less than that.
- 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:
  - 'bedrift'
  - weight loss after calving
  - weight before 'gold' period
  - 'huld'score
  - medical history including whether the cow has had ketosis before
  - fpf after calving (and related measures, as milk protein and milk fat)
  - fatty acids
  - calving number
  - weight difference before and after calving
  - 'ydelse' before calving

  - the length of the 'gold' period
  - birth complications
- 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 slightly better performance.
- Combining the performance enhancements mentioned above results in a prediction performance for a subset of 'bedrifter' which may be worth examining further.

4 Future Work

The following areas are interesting for future work:

4.1 Model

- A hyperparameter tuning may improve the prediction performance.
- Different models should be tested in search for better prediction performance, e.g. neural networks, random forests etc.

4.2 Data

- The data contains a substantial amount 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.
- Performance is significantly better for cows with at least 150 days of daily history before calving compared to cows with less than that. Thus, more data for each cow would benefit the model.
- As the data shows significant differences between 'bedrifter' due to e.g. management, it would be beneficial to further examine these differences and why they occur.
- If the model is to be used for new 'bedrifter' which are not included in the training data, it will be benficial to not include 'bedrift' as a feature.
- Allowing the model more data after calving may improve prediction performance for ketosis cases that happens further away from calving.
- 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.

5 Technical setup

5.1 Conda Environment

In this notebook the `py39_ketosis` conda environment is used.
It can be installed using the `py39_ketosis.yml` file.

5.2 Imports

[1]:
```
import re
import warnings

from catboost import CatBoostClassifier, Pool, EShapCalcType,␣
 ↪EFeaturesSelectionAlgorithm
import holoviews as hv
import hvplot.pandas
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from pandas.tseries.offsets import DateOffset
from pathlib import Path
from pprint import pprint as pp
import shap
from sklearn.metrics import classification_report, roc_auc_score, roc_curve,␣
 ↪precision_recall_curve, auc, precision_recall_fscore_support
from sklearn.model_selection import train_test_split, KFold
```

6 Exploratory analysis for a single bedrift

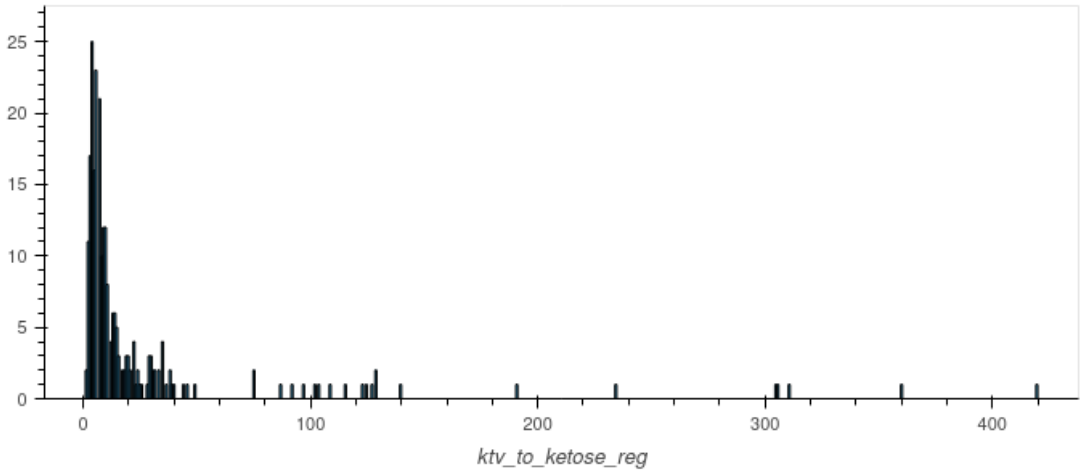
[2]:
```
data_folder = Path('/home/lasm/Downloads/AP1_data') # Local data folder
```

[3]:
```
df_hendelserprdag = pd.read_csv(data_folder/f'hendelserprdag{bedrift}.csv')
df_hendelserprdag.klvdato = pd.to_datetime(df_hendelserprdag.klvdato,␣
 ↪format='%d/%m/%y')
df_hendelserprdag.dato_start = pd.to_datetime(df_hendelserprdag.dato_start,␣
 ↪format='%d%b%y')
df_hendelserprdag.VISITDATETIME = pd.to_datetime(df_hendelserprdag.
 ↪VISITDATETIME, format='%d%b%Y:%H:%M:%S')
df_hendelserprdag.ENDVISITDATETIME = pd.to_datetime(df_hendelserprdag.
 ↪ENDVISITDATETIME, format='%d%b%Y:%H:%M:%S')

df_syg = pd.read_csv(data_folder/f'syg{bedrift}.csv', encoding='latin1')
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')
```

[4]:
```
# Plot distribution of time between calving and ketosis
df_syg = df_syg.assign(ktv_to_ketose_reg=df_syg.regdato - df_syg.klvdato)
df_syg[df_syg.navn == 'Ketose'].ktv_to_ketose_reg.dt.days.hvplot.hist(bins=500)
```

[4]:
```
:Histogram   [ktv_to_ketose_reg]   (ktv_to_ketose_reg_count)
```



[5]:
```
print(df_syg[df_syg.navn == 'Ketose'].overkat.unique())
print(df_syg[df_syg.navn == 'Ketose'].underkat.unique())
```

```
['STOFSKIFTELIDELSER hos kvæg']
['Ketose (Sygdomme)']
```

[6]:
```
print(df_hendelserprdag.iloc[:5,[3,6,9,14]].to_markdown(index=False))
```

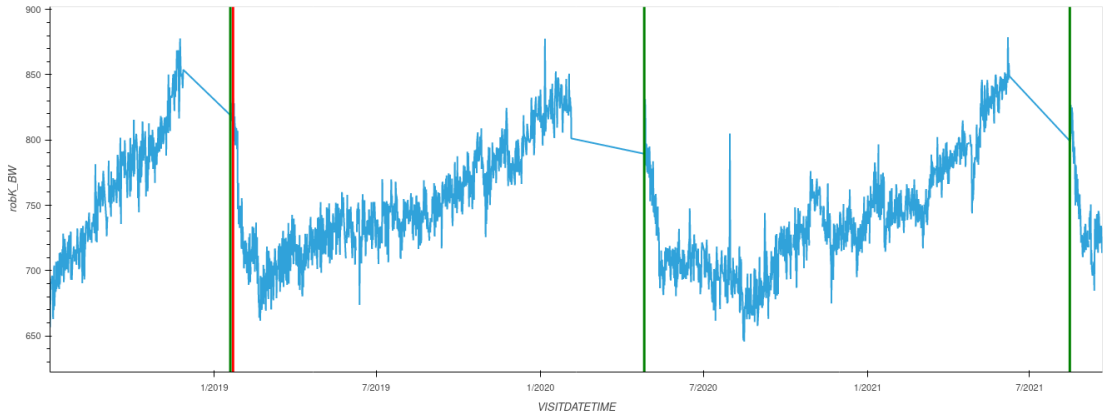
```
|   klvnr | VISITDATETIME       |   Ydelse_total |   robK_BW |
|--------:|:--------------------|---------------:|----------:|
|       7 | 2018-07-01 15:58:10 |             10 |   602.564 |
|       7 | 2018-07-02 00:25:26 |             15.9 |   622.542 |
|       7 | 2018-07-02 16:11:12 |             26.5 |   615.834 |
|       7 | 2018-07-03 03:01:14 |             17.3 |   612.734 |
|       7 | 2018-07-03 15:58:18 |             19.8 |   614.734 |
```

6.1 Visualize data before/after ketose

[7]:
```
df_ketose_dyr = df_syg[
    (df_syg.regdato > '2018-07-01') &
    (df_syg.navn == 'Ketose')]
```

[8]:
```
dyr_id = df_ketose_dyr.Dyr_Id.unique()[3]
df_ketose_dyr_obs = df_hendelserprdag[df_hendelserprdag.DYR_ID == dyr_id]
plots = []
plots.append(df_ketose_dyr_obs.hvplot(y=['robK_BW'], x='VISITDATETIME'))
plots.extend([hv.VLine(date).opts(color='red') for date in␣
 ↪df_ketose_dyr[df_ketose_dyr.Dyr_Id == dyr_id].regdato.unique()])
plots.extend([hv.VLine(date).opts(color='green') for date in df_ketose_dyr_obs.
 ↪klvdato.unique()])
hv.Overlay(plots).opts(height=500, width=1350)
```

[8]:
```
:Overlay
   .Curve.I   :Curve   [VISITDATETIME]   (robK_BW)
   .VLine.I   :VLine   [x,y]
   .VLine.II  :VLine   [x,y]
   .VLine.III :VLine   [x,y]
   .VLine.IV  :VLine   [x,y]
   .VLine.V   :VLine   [x,y]
```



7 Feature Engineering

7.1 Functions

[9]:
```
def create_target(df, unique_obs):
    targets = []
    unique_klvs = unique_obs[unique_obs.DYR_ID==df.DYR_ID.
 ↪iloc[0]]['klvdato_next'] # klvdato_next as these are the ones with obs.␣
 ↪before and after klv
    if len(unique_klvs)==0:
        return
    for kv_dato in unique_klvs:
        target = df[df.navn == 'Ketose'].regdato.between(pd.
 ↪to_datetime(kv_dato), pd.to_datetime(kv_dato) + DateOffset(days=24),␣
 ↪inclusive='both').any()
        dict_target = {'klvdato': kv_dato, 'target': target}
        targets.append(dict_target)

    df_target = pd.DataFrame(targets)
    return df_target


def create_features_before_calving(df):
    kv_dato = df.klvdato_next.iloc[0]
    df = df[df.VISITDATETIME.between(kv_dato - DateOffset(days=30*5), kv_dato,␣
 ↪inclusive='left')]
```

```
    df = df[['DYR_ID', 'klvdato_next', 'VISITDATETIME',
                'Ydelse_total','robK_BW', 'MILKFAT', 'MILKPROTEIN', 'fpf']].
 ↪resample('30D', on='VISITDATETIME', origin = kv_dato).agg(
                    {
                        'DYR_ID': 'first',
                        'klvdato_next': 'first',
                        'Ydelse_total': 'mean',
                        'robK_BW': 'mean',
                        'MILKFAT': 'mean',
                        'MILKPROTEIN': 'mean',
                        'fpf': 'mean',
                    })
    names =  {150: '_150_to_120_days_bf_klv', 120: '_120_to_90_days_bf_klv', 90:
 ↪ '_90_to_60_days_bf_klv', 60: '_60_to_30_days_bf_klv', 30:␣
 ↪'_30_to_0_days_bf_klv'}

    df['name'] = [names[(kv_dato-idx).days] for idx in df.index]
    df.index=[0]*len(df)
    df = df.pivot(index=None, columns='name', values=['Ydelse_total',␣
 ↪'robK_BW', 'MILKFAT', 'MILKPROTEIN','fpf'])
    df.columns = df.columns.map(''.join)

    return df


def create_gold_period_length(df):
    list_gold_period = []
    for kv_dato in df.klvdato.unique():
        last_date = df.VISITDATETIME[df.VISITDATETIME.between(pd.
 ↪to_datetime(kv_dato) - DateOffset(days=150), pd.to_datetime(kv_dato) -␣
 ↪DateOffset(days=24), inclusive='left')].max()
        if pd.isnull(last_date):
            continue
        gold_period_length = (kv_dato - last_date).days
        dict_gold_period = {'klvdato': kv_dato, 'gold_period_length':␣
 ↪gold_period_length}
        list_gold_period.append(dict_gold_period)

    df_gold_period = pd.DataFrame(list_gold_period)
    return df_gold_period


def create_syg_features(df, unique_obs):
    syg_features = []
```

```
    unique_klvs = unique_obs[unique_obs.DYR_ID==df.DYR_ID.
↪iloc[0]]['klvdato_next'] # klvdato_next as these are the ones with obs.␣
↪before and after klv
    if len(unique_klvs)==0:
        return
    for kv_dato in unique_klvs:
        tidl_ketose = len(df[(df.regdato <= kv_dato) & (df.navn=='Ketose')])

        temp = df[df.regdato.between(pd.to_datetime(kv_dato) -␣
↪DateOffset(days=150), pd.to_datetime(kv_dato) + DateOffset(days=10),␣
↪inclusive='both')]
        efterbyrd = len(temp[temp.navn == 'Efterbyrd'])
        maelkefeber = len(temp[temp.navn == 'Kælvningsfeber'])
        boerbetaend = len(temp[temp.navn == 'Børbetændelse'])
        dict_syg_features = {'klvdato': kv_dato, 'tidl_ketose': tidl_ketose,␣
↪'efterbyrd': efterbyrd, 'maelkefeber': maelkefeber, 'boer': boerbetaend}
        syg_features.append(dict_syg_features)

    df_syg_features = pd.DataFrame(syg_features)
    return df_syg_features


def create_kont_features(df, unique_obs):
    kont_features = []
    unique_klvs = unique_obs[unique_obs.DYR_ID==df.DYR_ID.iloc[0]][['klvdato',␣
↪'klvdato_next']] # klvdato_next as these are the ones with obs. before and␣
↪after klv
    if len(unique_klvs)==0:
        return
    for index, row in unique_klvs.iterrows():
        date_last_kvdato = row['klvdato']
        date_kvdato = row['klvdato_next']
        date_after_klv = row['klvdato_next'] + DateOffset(days=10)

        df_before_klv = df[df.kontdato.between(date_last_kvdato, date_kvdato,␣
↪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(date_kvdato, date_after_klv,␣
↪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'] = (date_kvdato - df_before_klv.kontdato.
↪min()).days
        first_kont_bf_klv.columns = [x + "_kont_first_bf_klv" 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'] = (date_kvdato - df_before_klv.kontdato.
↪max()).days
        last_kont_bf_klv.columns = [x + "_kont_last_bf_klv" 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() -␣
↪date_kvdato).days
        first_kont_after_klv.columns = [x + "_kont_first_after_klv" for x in␣
↪first_kont_after_klv.columns]

        kont_id = pd.DataFrame({'klvdato': date_kvdato}, 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
```

[10]:
```
def create_dataset(df_hendelserprdag, df_syg, df_kont, df_klv):

    # ------------------------------------------------
    #   Features from df_hendelserprdag
```

```
    # ------------------------------------------------
    # Create fpf feature
    df_hendelserprdag['fpf'] = df_hendelserprdag['MILKFAT'] /␣
↪df_hendelserprdag['MILKPROTEIN']

    # Idx cols
    idx_cols = ['DYR_ID', 'klvdato']

    df_hendelserprdag = df_hendelserprdag.dropna(subset=idx_cols)
    # Sort by date
    df_hendelserprdag = df_hendelserprdag.sort_values(by=idx_cols +␣
↪['VISITDATETIME'])

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

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

    if len(unique_obs)==0:
        return

    # -------------------------
    #   Features before calving
    # -------------------------
    df_bf_klv_features = df_hendelserprdag.groupby(['DYR_ID','klvdato_next']).
↪apply(create_features_before_calving).reset_index(level=2, drop=True).
↪reset_index()
    df_bf_klv_features.rename(columns = {'klvdato_next': 'klvdato'}, inplace =␣
↪True)

    # Relative features before calving
    relative_features = ['Ydelse_total','robK_BW', 'MILKFAT', 'MILKPROTEIN',␣
↪'fpf']
    needed_columns = [name + '_150_to_120_days_bf_klv' for name in␣
↪relative_features]
    missing_columns = [column for column in needed_columns if column not in␣
↪df_bf_klv_features.columns]
    df_bf_klv_features[missing_columns] = np.nan
    bf_klv_features = []
```

```
    for feature in relative_features:
        temp_relative = df_bf_klv_features[[col for col in df_bf_klv_features.
↪columns if col.startswith(feature)]] /␣
↪df_bf_klv_features[[feature+'_150_to_120_days_bf_klv']].to_numpy()
        temp_relative.drop(feature+'_150_to_120_days_bf_klv', axis=1,␣
↪inplace=True)
        temp_relative.columns = [x + "_relative" for x in temp_relative.columns]
        bf_klv_features.append(temp_relative)
    df_bf_klv_features_relative = pd.concat(bf_klv_features, axis=1)

    df_bf_klv_features_full = df_bf_klv_features.
↪merge(df_bf_klv_features_relative, how='outer', right_index=True,␣
↪left_index=True)

    # -------------------------
    #   Features after calving
    # -------------------------
    no_obs_aft_klv = 10
    df_after_klv_features = df_hendelserprdag.groupby(idx_cols).
↪head(no_obs_aft_klv)

    # TODO: may add the other time features:
    # Address, time in robot,
    shift_cols = ['Ydelse_total','robK_BW','MILKPROTEIN','MILKFAT','fpf']
    for shift_no in range(1, no_obs_aft_klv):
        shifted = df_after_klv_features.groupby(idx_cols)[shift_cols].
↪shift(shift_no)
        shifted.columns = [x + "_obs_" + str(no_obs_aft_klv - shift_no) +␣
↪"_aft_klv" for x in shift_cols]
        df_after_klv_features = pd.concat((df_after_klv_features, shifted),␣
↪axis=1)

    df_after_klv_features.rename(columns = {name:␣
↪name+f'_obs_{no_obs_aft_klv}_aft_klv' for name in shift_cols}, inplace =␣
↪True)

    filter_col = [col for col in df_after_klv_features if col.
↪startswith(tuple(shift_cols))]
    df_after_klv_features = df_after_klv_features.groupby(idx_cols).tail(1).
↪loc[:, [
            'DYR_ID', 'klvdato'] + filter_col].reset_index(drop=True)

    # Relative features after calving
    relative_features = ['Ydelse_total','robK_BW', 'MILKFAT', 'MILKPROTEIN',␣
↪'fpf']
    aft_klv_features = []
```

```
    for feature in relative_features:
        temp_relative = df_after_klv_features[[col for col in␣
↪df_after_klv_features.columns if col.startswith(feature)]] /␣
↪df_after_klv_features[[feature+'_obs_1_aft_klv']].to_numpy()
        temp_relative.drop(feature+'_obs_1_aft_klv', axis=1, inplace=True)
        temp_relative.columns = [x + "_relative" for x in temp_relative.columns]
        aft_klv_features.append(temp_relative)
    df_after_klv_features_relative = pd.concat(aft_klv_features, axis=1)

    df_after_klv_features_full = df_after_klv_features.
↪merge(df_after_klv_features_relative, how='outer', right_index=True,␣
↪left_index=True)

    # 'Static' features
    df_static_features = df_hendelserprdag.groupby(idx_cols).
↪head(no_obs_aft_klv).groupby(idx_cols).agg(
        {'bedrift':'first', 'race_id':'first', 'klvnr':'first', 'MILKSPEEDMAX':
↪'mean',
        'MILKTIMEKO':'mean', 'LFDEADMILKTIME':'mean', 'MILKTEMPERATURE': 'mean',
        'RFDEADMILKTIME':'mean','LRDEADMILKTIME':'mean', 'RRDEADMILKTIME':
↪'mean'}).reset_index()
    df_static_features

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

    # ------------------------------------------------
    #  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)

    # ------------------------------------------------
    #  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:
        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.merge(
        df_after_klv_features_full, 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=idx_cols)

    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=idx_cols)

    if df_kont_features is not None:
        df_full_features = df_full_features.merge(
            df_kont_features, how='left', on=idx_cols)

    # ------------------------------------------------
    #  Create final additional features
    # ------------------------------------------------
    needed_columns = ['robK_BW_30_to_0_days_bf_klv',␣
↪'robK_BW_60_to_30_days_bf_klv', 'robK_BW_90_to_60_days_bf_klv',␣
↪'robK_BW_120_to_90_days_bf_klv']
    missing_columns = [column for column in needed_columns if column not in␣
↪df_full_features.columns]
    df_full_features[missing_columns] = np.nan
    # Weight difference between last available 30 day mean of weight and the␣
↪first datapoint after calving
    df_full_features['klv_weight_diff'] = np.
↪where(~df_full_features['robK_BW_30_to_0_days_bf_klv'].isnull(),
                                                                    ␣
↪df_full_features['robK_BW_30_to_0_days_bf_klv']-df_full_features['robK_BW_obs_1_aft_klv'],
```

```
                                                    np.
↪where(~df_full_features['robK_BW_60_to_30_days_bf_klv'].isnull(),
                                                                ␣
↪df_full_features['robK_BW_60_to_30_days_bf_klv']-df_full_features['robK_BW_obs_1_aft_klv'],
                                                     np.
↪where(~df_full_features['robK_BW_90_to_60_days_bf_klv'].isnull(),
                                                                ␣
↪df_full_features['robK_BW_90_to_60_days_bf_klv']-df_full_features['robK_BW_obs_1_aft_klv'],
                                                                          ␣
↪df_full_features['robK_BW_120_to_90_days_bf_klv']-df_full_features['robK_BW_obs_1_aft_klv']

    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
```

7.2 Create dataset

[11]:
```
hendelserprdag_files = [i for i in data_folder.glob('hendelserprdag*.csv')]
bedrifts = [re.findall('\d{2,}', str(file))[0] for file in hendelserprdag_files]
```

[12]:
```
final_dfs = []
for idx, bedrift in enumerate(bedrifts):
    # 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_hendelserprdag = df_hendelserprdag[df_hendelserprdag.klvnr > 1]
    df_hendelserprdag = df_hendelserprdag[
        (df_hendelserprdag.Ydelse_total.between(0.5, 100)) &
        (df_hendelserprdag.robK_BW.between(100, 1_200))]

    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 = df_syg[['DYR_ID', 'regdato','klvdato', 'navn']]
    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']]

    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)

    final_dfs.append(df_final)
    #print(f'Loaded {idx+1} of {len(bedrifts)}')

df_final_full = pd.concat(final_dfs)
```

```
Columns (16) have mixed types.Specify dtype option on import or set
low_memory=False.
Columns (16) have mixed types.Specify dtype option on import or set
low_memory=False.
Columns (16) have mixed types.Specify dtype option on import or set
low_memory=False.
```

```
Columns (16) have mixed types.Specify dtype option on import or set
low_memory=False.
Columns (16) have mixed types.Specify dtype option on import or set
low_memory=False.
```

[13]:
```
df_final_full.to_csv(data_folder/'df_final_full.csv', index = False)
df_final_full.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 26829 entries, 0 to 8
Columns: 190 entries, DYR_ID to target
dtypes: bool(1), datetime64[ns](1), float64(185), object(3)
memory usage: 38.9+ MB
```

8 Train model across all 'bedrifter'

8.1 Load data

[12]:
```
df_full_dataset = pd.read_csv(data_folder/'df_final_full.csv')
df_full_dataset = df_full_dataset.drop(['DYR_ID','klvdato'], axis=1)
df_full_dataset['bedrift'] = df_full_dataset['bedrift'].astype(str)
```

[13]:
```
df_full_dataset.dtypes
```

[13]:
```
Ydelse_total_120_to_90_days_bf_klv         float64
Ydelse_total_60_to_30_days_bf_klv          float64
Ydelse_total_90_to_60_days_bf_klv          float64
robK_BW_120_to_90_days_bf_klv              float64
robK_BW_60_to_30_days_bf_klv               float64
                                             …
FA_Preformed_i_TFA_kont_first_after_klv    float64
days_before_kont_first_after_klv           float64
klv_weight_diff                            float64
month_of_year                                int64
target                                        bool
Length: 188, dtype: object
```

8.2 Split data

[14]:
```
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 ketosis cases: {round(len(y_test[y_test==1]) /␣
 ↪len(y_test), 2)}')
```

```
Length of test dataset: 6708
Percentage of ketosis cases: 0.06
```

8.3 Train model

[15]:
```
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
)
```

[15]:
```
<catboost.core.CatBoostClassifier at 0x7f0fb1a44640>
```

8.4 Performance

[16]:
```
y_pred_prob = model.predict_proba(X_test, )[:,1]
y_pred = y_pred_prob >= 0.5
```

8.4.1 Classification report

[17]:
```
pp(classification_report(y_true=y_test.values,y_pred=y_pred))
```

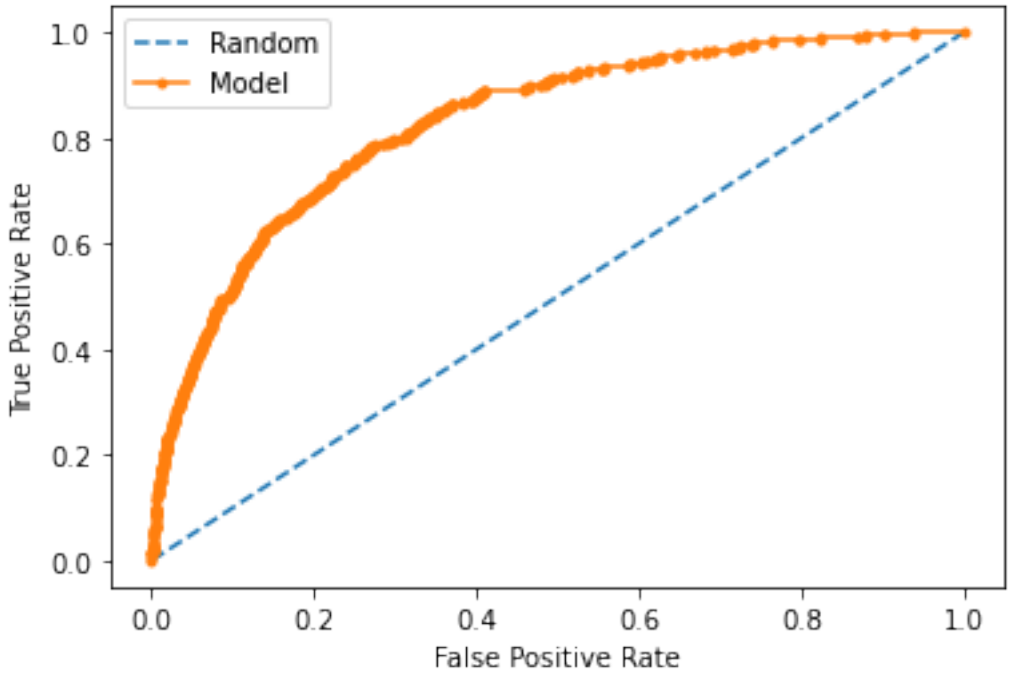
```
('              precision    recall  f1-score   support\n'
 '\n'
 '       False       0.97      0.93      0.95      6337\n'
 '        True       0.26      0.44      0.33       371\n'
 '\n'
 '    accuracy                           0.90      6708\n'
 '   macro avg       0.61      0.68      0.64      6708\n'
 'weighted avg       0.93      0.90      0.91      6708\n')
```

8.4.2 ROC curve

[18]:
```
# Area under ROC curve
roc_auc_score(y_score=y_pred_prob, y_true=y_test)
```

[18]:
```
0.8306106225066748
```

[19]:
```
# 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()
```

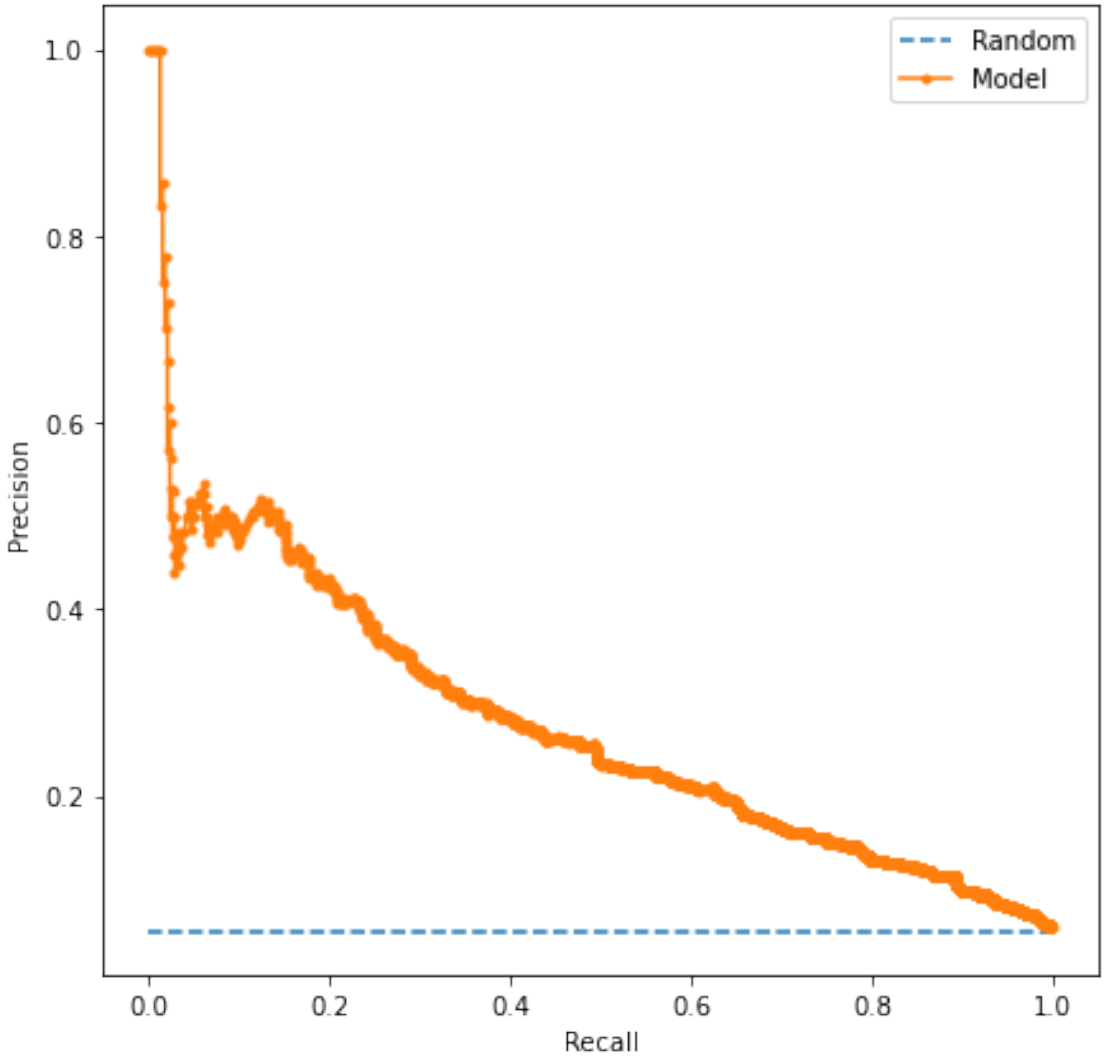


8.4.3 Precision-Recall Curve

[20]:
```
# Precision-recall curve
precision, recall, thresholds = precision_recall_curve(y_test, y_pred_prob)
# Area under precision-recall curve
auc(recall, precision)
```

[20]: 0.27733320831188873

[21]:
```
# 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()
```



8.5 Classification inspection

[22]:
```
def measures(df):
    no_obs = int(len(df))
    no_ketose = 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, no_ketose, pr_auc, roc_auc, precision, recall,␣
↪fscore), index=['no_obs', 'no_ketose', 'PR_AUC', 'ROC_AUC', 'precision',␣
↪'recall', 'f1'])
```

8.5.1 Performance across 'bedrifter'

[23]:
```
test_set = X_test.merge(y_test, how='inner', left_index=True, right_index=True)
test_set['pred']=y_pred
test_set['pred_prob']=y_pred_prob
```

[24]:
```
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 |   no_ketose |   PR_AUC |   ROC_AUC |   precision |
|---------:|------------:|---------:|----------:|------------:|
|       45 |           1 |   1      |  1        |    1        |
|       33 |           1 |   1      |  1        |    0        |
|       43 |           1 |   1      |  1        |    0        |
|       69 |           1 |   1      |  1        |    1        |
|       15 |           1 |   1      |  1        |    0        |
|       39 |           1 |   1      |  1        |    0.333333 |
```

33	1	1	1	0	
5	1	1	1	0	
28	2	1	1	0	
10	2	0.7917	0.9375	0.5	
69	11	0.6599	0.835423	0.466667	
54	11	0.6107	0.854123	0.470588	
51	11	0.5634	0.8	0.294118	
41	12	0.5552	0.689655	0.4	
47	4	0.5364	0.918605	0.235294	
74	2	0.5271	0.819444	0.333333	
83	10	0.5167	0.843836	0.227273	
54	5	0.5057	0.893878	1	
61	6	0.5003	0.869697	1	
188	42	0.4785	0.777071	0.291045	
126	9	0.4303	0.785375	0.428571	
45	5	0.4143	0.875	0.263158	
124	6	0.4094	0.637006	0.666667	
93	8	0.4092	0.767647	0.152174	
57	4	0.3871	0.773585	0	
63	4	0.3752	0.847458	0.5	
21	4	0.3656	0.691176	0.428571	
15	4	0.3621	0.704545	0.333333	
48	7	0.3492	0.721254	0.230769	
88	7	0.3458	0.761905	0.5	
74	10	0.3412	0.673438	0.333333	
89	6	0.3315	0.793173	0.333333	
62	7	0.2835	0.787013	0	
67	13	0.2759	0.636752	0.263158	
105	8	0.2605	0.731959	0.190476	
46	4	0.2577	0.875	0	
84	4	0.2576	0.83125	0.230769	
44	1	0.25	0.976744	0	
32	1	0.25	0.967742	0	
41	1	0.25	0.975	0	

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

8.5.2 Performance for cows with 150 days of daily history before calving compared to less than that

[25]:
```
relative_features = ['Ydelse_total','robK_BW', 'MILKFAT', 'MILKPROTEIN', 'fpf']
missing_values_cols = [name + '_150_to_120_days_bf_klv' for name in␣
 ↪relative_features]
test_set['missing'] = test_set[missing_values_cols].isnull().any(axis=1).values
```

[26]:
```
missing_compare = test_set.groupby('missing').apply(measures).reset_index()
pp(missing_compare.sort_values(by='PR_AUC',ascending=False))
```

	missing	no_obs	no_ketose	PR_AUC	ROC_AUC	precision	recall	f1
0	False	4774.0	270.0	0.2994	0.833662	0.261856	0.470370	0.336424
1	True	1934.0	101.0	0.2076	0.822808	0.264706	0.356436	0.303797

As can be seen, performance is much greater for cows with 150 days of daily history before calving.

9 Train model using only 'bedrifter' which are similar in their management

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

[27]:
```
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 = 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
    )

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

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

    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
```

[28]:
```
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	no_ketose	PR_AUC	ROC_AUC	precision	recall
43	0.25	1	1	1	1
17.75	0.25	1	1	0	0
7	0.25	1	1	0	0
46.5	0.25	1	1	0	0
18.25	0.25	1	1	0	0
65.5	0.25	1	1	1	1
8	1.5	0.869467	0.898148	0.466667	0.888889
3	1	0.85415	0.833333	0.75	1
15.25	3.25	0.6763	0.778922	0.289286	0.9375
29.75	1.25	0.67125	0.815278	0	0
44	7.75	0.573575	0.855702	0.311465	0.86453
39.5	0.75	0.5423	0.842105	0	0
57	4	0.542233	0.866067	0.444444	0.380952
23.5	0.5	0.525	0.804348	0	0
53.5	11.5	0.476275	0.733074	0.298514	0.853846
32.5	3	0.47495	0.826296	0.3125	0.5
49.25	8	0.473625	0.824134	0.226846	0.936508
60.75	2.75	0.467225	0.874533	0.375	0.175
30.75	2.25	0.463975	0.866404	0.425	0.5625
59.25	5	0.45015	0.761818	0.369444	0.530357
61.75	3.75	0.442767	0.839369	0.40873	0.431746
42.5	1	0.4403	0.88184	0.333333	0.666667
199.75	46.5	0.4383	0.709918	0.289814	0.844111
53.25	8.5	0.411075	0.76999	0.276786	0.676136
50.75	6.5	0.4082	0.727672	0.239621	0.61875
28.5	3.75	0.39915	0.806729	0.305618	0.733333
27	1.25	0.3972	0.932197	0	0
31.25	2	0.388475	0.838292	0.0625	0.25
43	1	0.3794	0.842931	0.333333	0.333333
44.25	2.25	0.378667	0.800854	0.333333	0.166667
56.75	10.75	0.3764	0.65712	0.30135	0.474398
80.5	9.25	0.37375	0.770726	0.315336	0.562879
52	5.25	0.3649	0.754195	0.227083	0.445833
83	8.25	0.339475	0.806794	0.204759	0.874008

```
|    44.75 |        6    | 0.335725 |  0.726226 |    0.262171 | 0.638095 |
|    57    |        2.75 | 0.325775 |  0.800701 |    0.5      | 0.1875   |
|    55.25 |        0.75 | 0.3244   |  0.867932 |    0.5      | 0.25     |
|    40.5  |        1.25 | 0.321    |  0.847613 |    0.25     | 0.25     |
|    45.25 |        3.25 | 0.3154   |  0.795744 |    0.246528 | 0.5625   |
|    99.5  |        5    | 0.31105  |  0.775927 |    0.25     | 0.05     |
```

[29]:
```
best_bedrifter = compare_bedrifter[compare_bedrifter['PR_AUC']>=0.4]['bedrift'].
 ↪values # 0.4 is an example of a cutoff
```

[30]:
```
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 ketosis cases: {round(len(y_test[y_test==1]) /␣
 ↪len(y_test), 2)}')
```

```
Length of test dataset: 889
Percentage of ketosis cases: 0.12
```

[31]:
```
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
)
```

[31]:
```
<catboost.core.CatBoostClassifier at 0x7f0faa3ba6d0>
```

9.1 Performance

As can be seen in the following, performance is greatly improved.

[32]:
```
y_pred_prob = model.predict_proba(X_test, )[:,1]
y_pred = y_pred_prob >= 0.5
```

9.1.1 Classification report

[33]:
```
pp(classification_report(y_true=y_test.values,y_pred=y_pred))
```

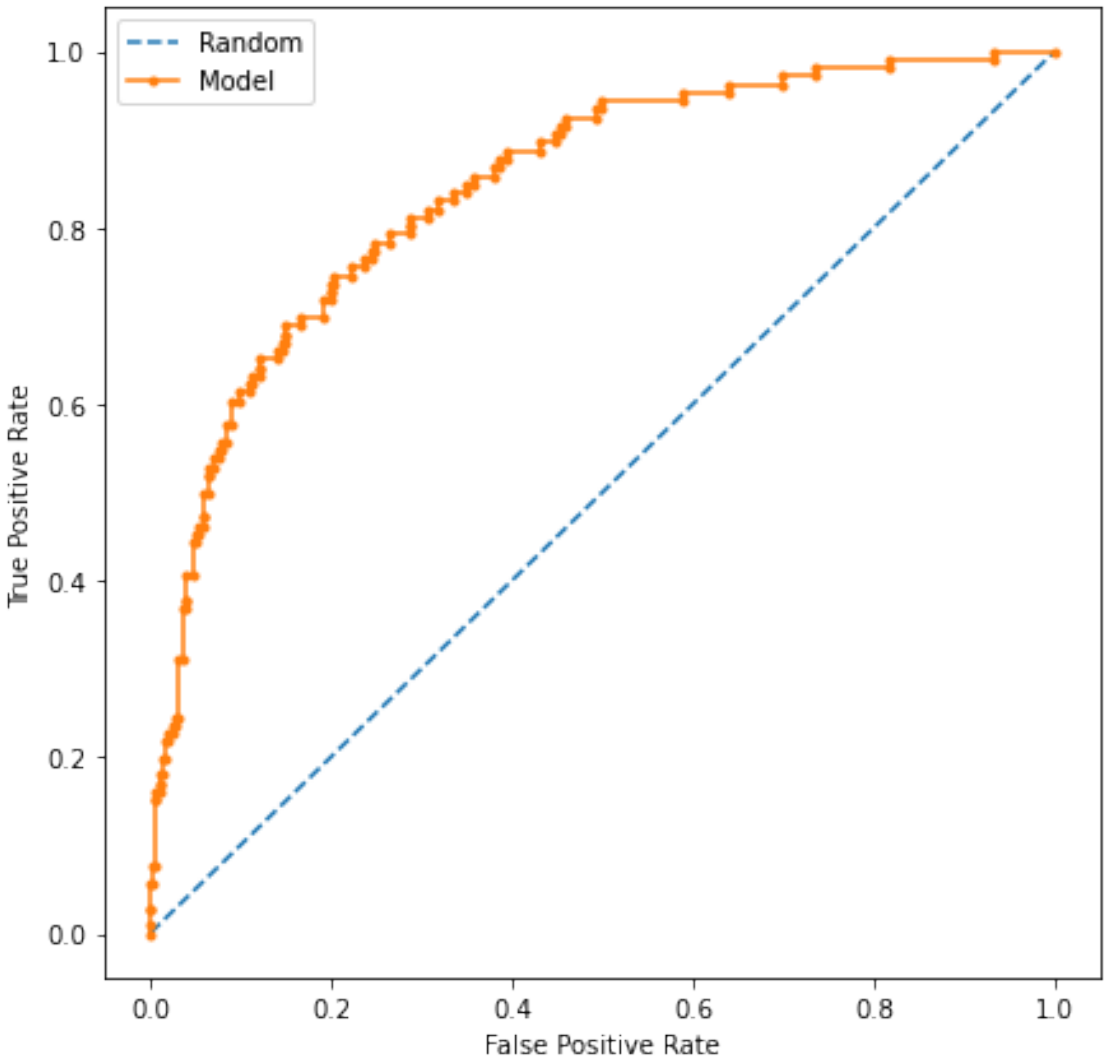
```
('              precision    recall  f1-score   support\n'
 '\n'
 '       False       0.96      0.77      0.85       783\n'
 '        True       0.30      0.75      0.43       106\n'
 '\n'
 '    accuracy                           0.76       889\n'
 '   macro avg       0.63      0.76      0.64       889\n'
 'weighted avg       0.88      0.76      0.80       889\n')
```

9.1.2 ROC curve

[34]:
```
# Area under ROC curve
roc_auc_score(y_score=y_pred_prob, y_true=y_test)
```

[34]:
```
0.8458517072700547
```

[35]:
```
# 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()
```



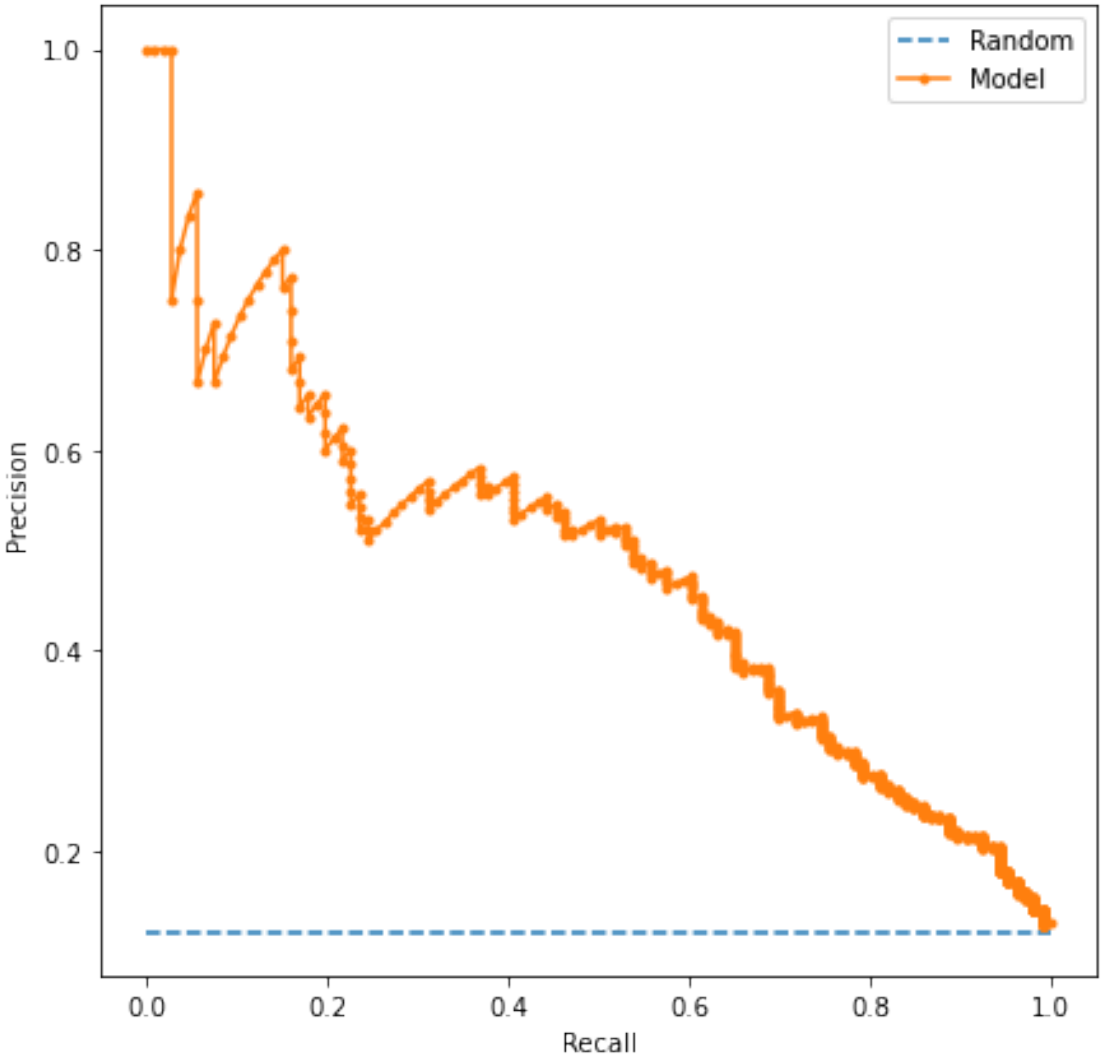
9.1.3 Precision-Recall Curve

[36]:
```
# Precision-recall curve
precision, recall, thresholds = precision_recall_curve(y_test, y_pred_prob)
# Area under precision-recall curve
auc(recall, precision)
```

[36]: 0.4822341148220858

[37]:
```
# 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()
# As can be seen, there seems to be strong signal for a small subset of all␣
 ↪ketosis cases.
```



9.1.4 Performance for cows with 150 days of daily history before calving compared to less than that

[38]:
```
test_set = X_test.merge(y_test, how='inner', left_index=True, right_index=True)
test_set['pred']=y_pred
test_set['pred_prob']=y_pred_prob
relative_features = ['Ydelse_total','robK_BW', 'MILKFAT', 'MILKPROTEIN', 'fpf']
```

```
missing_values_cols = [name + '_150_to_120_days_bf_klv' for name in␣
↪relative_features]
test_set['missing'] = test_set[missing_values_cols].isnull().any(axis=1).values
```

[39]:
```
missing_compare = test_set.groupby('missing').apply(measures).reset_index()
pp(missing_compare.sort_values(by='PR_AUC',ascending=False))
```

	missing	no_obs	no_ketose	PR_AUC	ROC_AUC	precision	recall	f1
0	False	724.0	92.0	0.5146	0.858799	0.334862	0.793478	0.470968
1	True	165.0	14.0	0.2131	0.750710	0.155556	0.500000	0.237288

As can be seen, performance is again much greater for cows with 150 days of daily history before calving.

10 Feature Importance

Looking at the most important features, it is found that it complies with what was hypothesized by domain experts. The most important features include: - 'bedrift' - weight loss after calving - weight before 'gold' period - 'huld'score - medical history including whether the cow has had ketosis before - fpf after calving (and related measures, as milk protein and milk fat) - fatty acids - calving number - weight difference before and after calving - 'ydelse' before calving - the length of the 'gold' period - birth complications

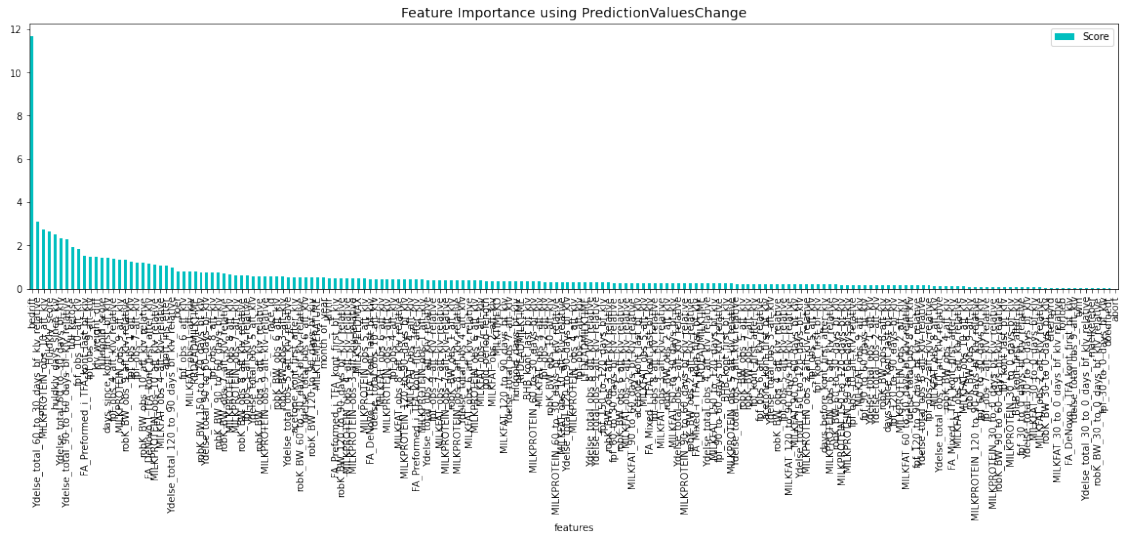
[40]:
```
shap_values = model.get_feature_importance(Pool(X_test,␣
↪label=y_test,cat_features=cat_col_indices),
                                                                    ␣
↪type="ShapValues")
shap.summary_plot(shap_values[:,:-1], X_test, max_display=60)
```


bedrift
Huldklv_score
MILKPROTEIN_obs_10_aft_klv
huldklv_days_since_klv
Ydelse_total_60_to_30_days_bf_klv
Ydelse_total_60_to_30_days_bf_klv_relative
Ydelse_total_90_to_60_days_bf_klv_relative
fpf_obs_10_aft_klv
tidl_ketose
fpf_obs_9_aft_klv
robK_BW_obs_9_aft_klv_relative
days_since_kont_first_bf_klv
klv_weight_diff
FA_Preformed_i_TFA_kont_last_bf_klv
fpf_obs_1_aft_klv
MILKPROTEIN_obs_9_aft_klv
MILKPROTEIN_obs_10_aft_klv_relative
abort_ins_efter
fpf_obs_4_aft_klv_relative
FA_DeNovo_i_TFA_kont_first_after_klv
fpf_obs_7_aft_klv
huldgold_score
robK_BW_obs_10_aft_klv_relative
fpf_obs_6_aft_klv
MILKFAT_obs_4_aft_klv_relative
Ydelse_total_90_to_60_days_bf_klv
robK_BW_150_to_120_days_bf_klv
MILKFAT_obs_1_aft_klv
fpf_obs_8_aft_klv
robK_BW_obs_7_aft_klv_relative
MILKPROTEIN_obs_9_aft_klv_relative
robK_BW_90_to_60_days_bf_klv
boer
robK_BW_obs_8_aft_klv_relative
MILKPROTEIN_obs_8_aft_klv_relative
klvnr
Ydelse_total_obs_5_aft_klv
MILKPROTEIN_obs_3_aft_klv
RRDEADMILKTIME
fpf_obs_3_aft_klv_relative
robK_BW_120_to_90_days_bf_klv
Ydelse_total_120_to_90_days_bf_klv_relative
MILKPROTEIN_obs_8_aft_klv
Ydelse_total_obs_10_aft_klv
Ydelse_total_obs_6_aft_klv
MILKPROTEIN_obs_4_aft_klv_relative
MILKPROTEIN_obs_3_aft_klv_relative
robK_BW_obs_3_aft_klv
MILKFAT_obs_6_aft_klv
MILKFAT_obs_8_aft_klv_relative
FA_DeNovo_i_TFA_kont_last_bf_klv
fpf_obs_9_aft_klv_relative
MILKFAT_obs_7_aft_klv_relative
huldgold_days_bf_klv
MILKPROTEIN_obs_5_aft_klv
robK_BW_obs_6_aft_klv
MILKPROTEIN_obs_7_aft_klv
MILKFAT_obs_7_aft_klv
MILKPROTEIN_obs_7_aft_klv_relative
MILKPROTEIN_obs_6_aft_klv_relative
−0.8
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
SHAP value (impact on model output)
High
Feature value
Low

[41]:
```
fi = model.get_feature_importance(Pool(X_test,␣
↪label=y_test,cat_features=cat_col_indices),
                                                                    ␣
↪type='PredictionValuesChange')
feature_score = pd.DataFrame(list(zip(X_test.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()
```



11 Feature elimination

Due to the abundance of features, correlations between the different features are first examined.

[42]:
```
# 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)
```

[42]:
```
66
```

Due to many features being highly correlated, and many features having little to no effect on the target value when looking at the feature importance above, feature elimination is tried.

[43]:
```
cat_col_indices = np.where(X_train.dtypes == object)[0]

model = CatBoostClassifier(
    class_weights=(1,10),
    n_estimators=500
)

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

summary = model.select_features(
    train_pool,
    eval_set=test_pool,
    features_for_select='0-186',
    num_features_to_select=20,
    steps=20,
    algorithm=EFeaturesSelectionAlgorithm.RecursiveByShapValues,
    shap_calc_type=EShapCalcType.Regular,
    train_final_model=True,
    logging_level='Silent')
```

[44]:
```
selected_features = summary['selected_features_names']
selected_features
```

[44]:
```
['Ydelse_total_30_to_0_days_bf_klv',
 'MILKPROTEIN_obs_10_aft_klv',
 'fpf_obs_10_aft_klv',
 'MILKPROTEIN_obs_9_aft_klv',
 'MILKPROTEIN_obs_7_aft_klv',
 'MILKPROTEIN_obs_2_aft_klv',
 'MILKFAT_obs_1_aft_klv',
 'robK_BW_obs_10_aft_klv_relative',
 'MILKPROTEIN_obs_10_aft_klv_relative',
 'MILKPROTEIN_obs_9_aft_klv_relative',
 'MILKPROTEIN_obs_5_aft_klv_relative',
 'bedrift',
 'abort_ins_efter',
 'doedfoedt',
```

```
 'Huldklv_score',
 'huldgold_score',
 'huldklv_days_since_klv',
 'tidl_ketose',
 'boer',
 'FA_Mixed_i_TFA_kont_first_after_klv']
```

[45]:
```
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
)
```

[45]:
```
<catboost.core.CatBoostClassifier at 0x7f0fa6ceaee0>
```

11.1 Performance

As can be seen in the following, by removing 150+ features, performance is actually improved.

[46]:
```
y_pred_prob = model.predict_proba(X_test_subset, )[:,1]
y_pred = y_pred_prob >= 0.5
```

11.1.1 Classification report

[47]:
```
pp(classification_report(y_true=y_test.values,y_pred=y_pred))
```

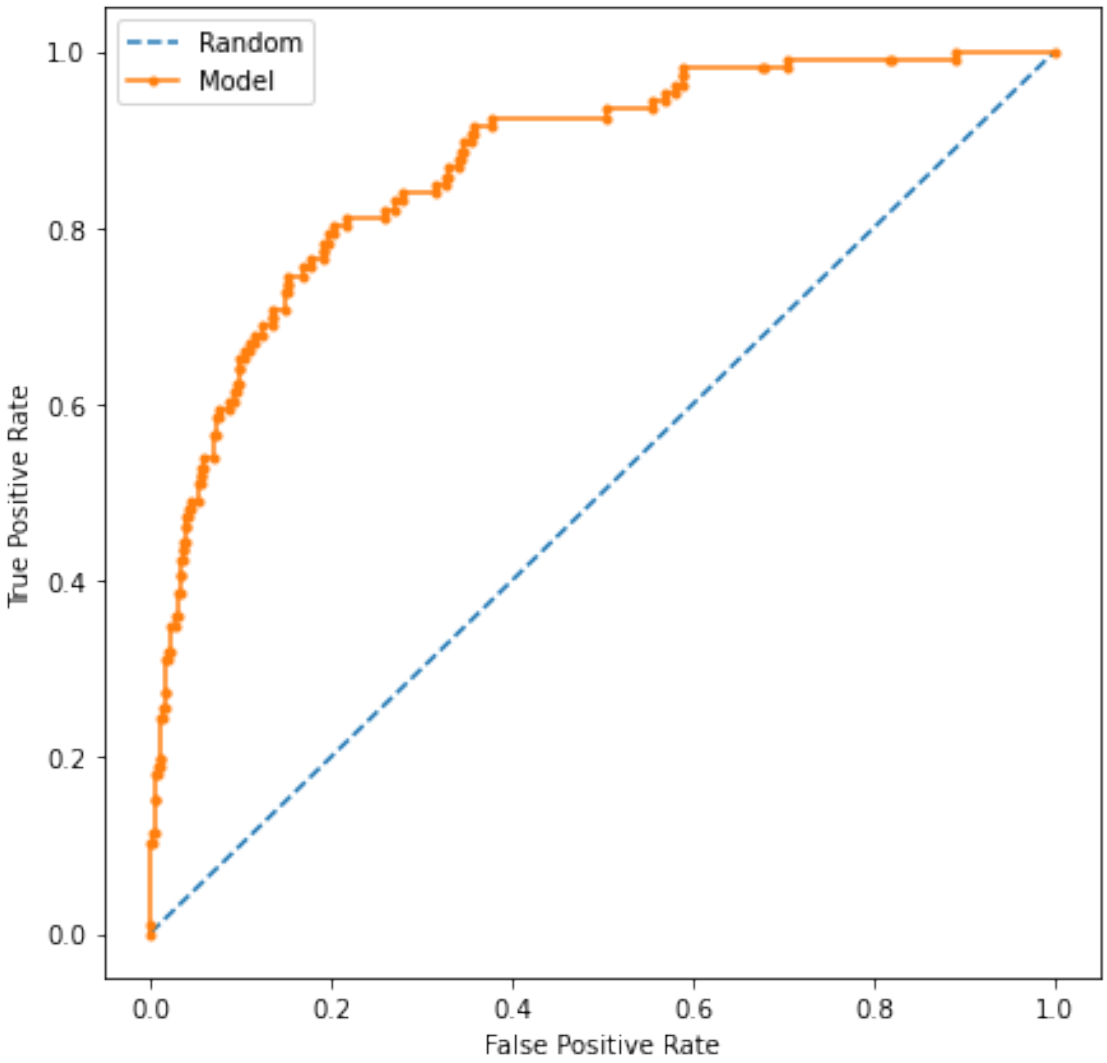
```
('              precision    recall  f1-score   support\n'
 '\n'
 '       False       0.97      0.76      0.85       783\n'
 '        True       0.31      0.81      0.45       106\n'
 '\n'
 '    accuracy                           0.76       889\n'
 '   macro avg       0.64      0.78      0.65       889\n'
 'weighted avg       0.89      0.76      0.80       889\n')
```

11.1.2 ROC curve

[48]:
```
# Area under ROC curve
roc_auc_score(y_score=y_pred_prob, y_true=y_test)
```

[48]:
```
0.8706474854815779
```

[50]:
```
# 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()
```



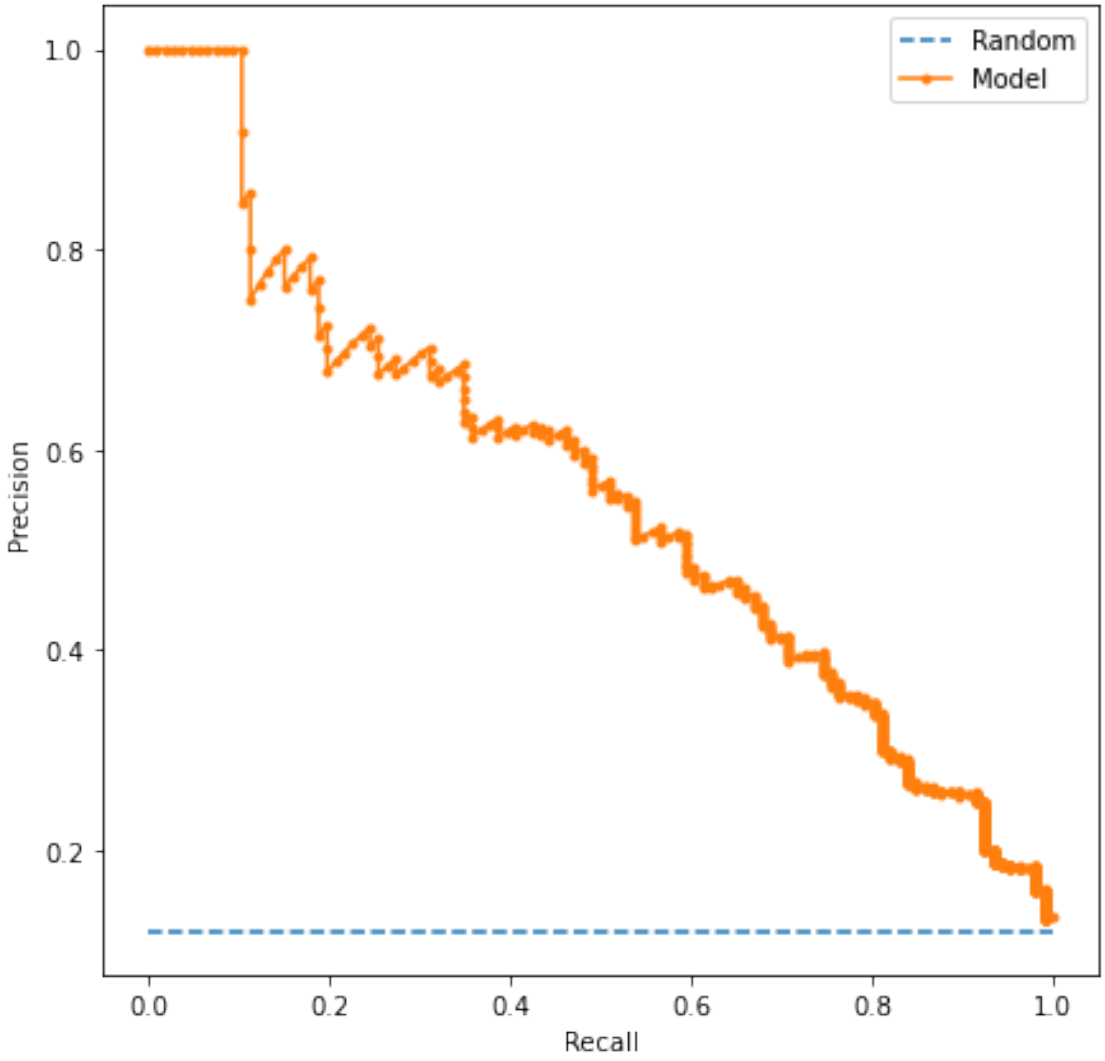
11.1.3 Precision-Recall Curve

[51]:
```
# Precision-recall curve
precision, recall, thresholds = precision_recall_curve(y_test, y_pred_prob)
# Area under precision-recall curve
auc(recall, precision)
```

[51]: 0.5564642787499662

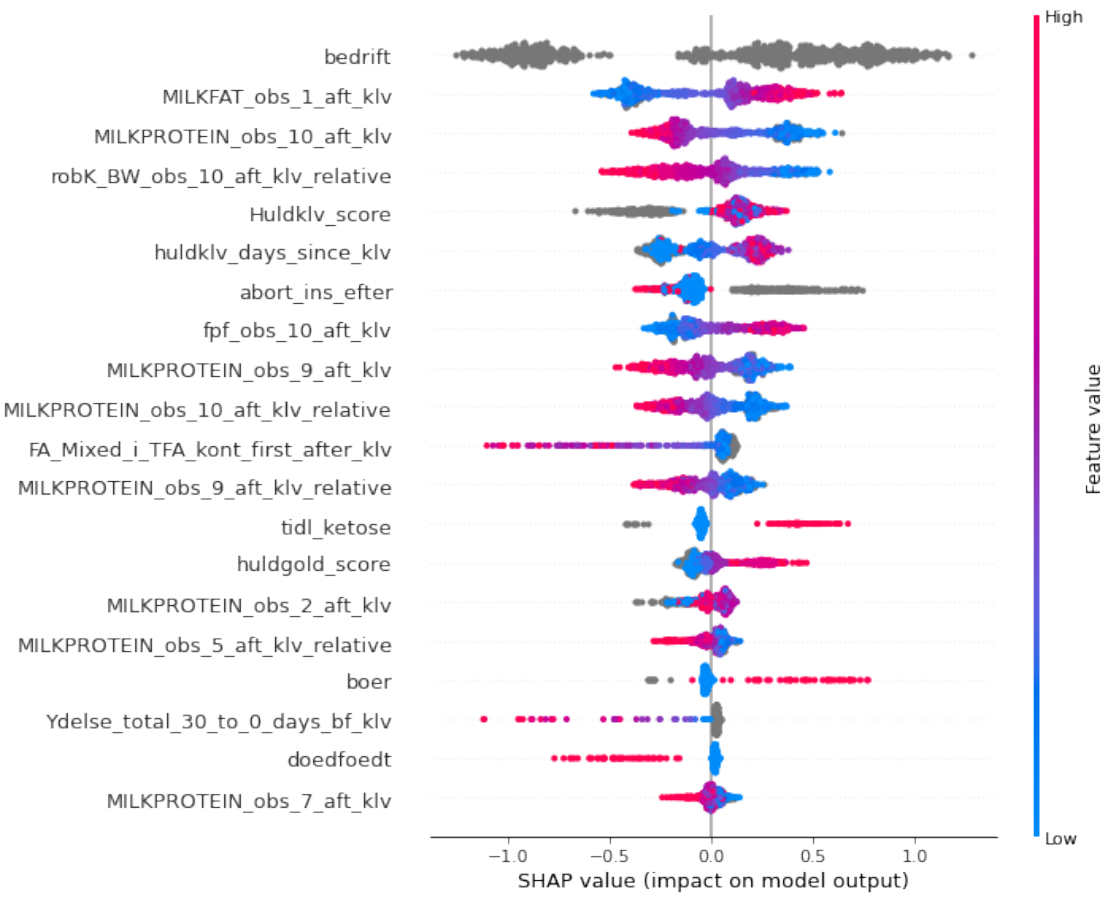
[52]:
```
# Plot precision-recall curve
random = len(y_test[y_test==1]) / len(y_test)
plt.plot([0, 1], [random, random], linestyle='--', label='Random')
plt.plot(recall, precision, marker='.', label='Model')
plt.xlabel('Recall')
```

```
plt.ylabel('Precision')
plt.legend()
plt.rcParams["figure.figsize"] = (7,7)
plt.show()
```



Looking at the feature importance, the effects are similar:

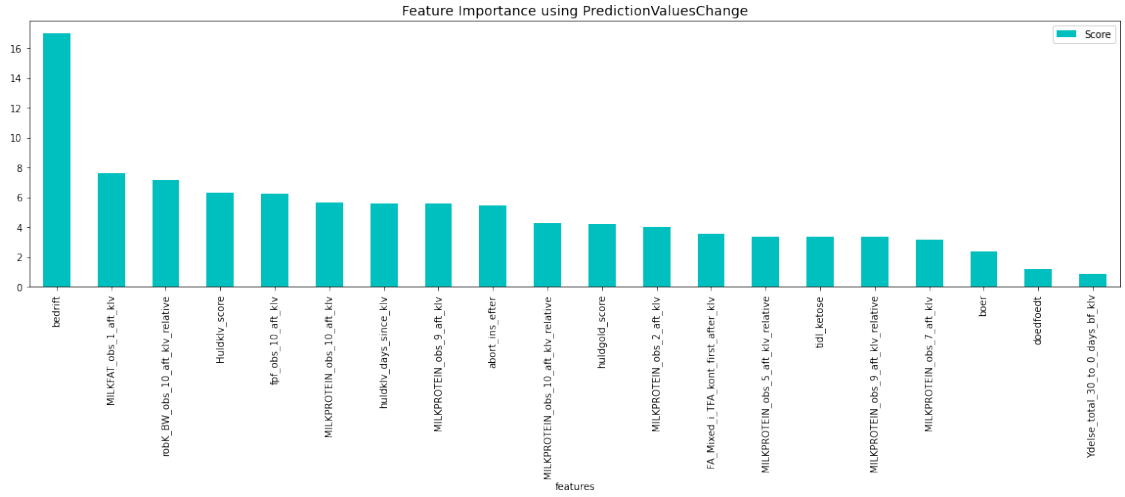
[53]:
```
shap_values = model.get_feature_importance(Pool(X_test_subset,␣
↪label=y_test,cat_features=cat_col_indices),
                                                                ␣
↪type="ShapValues")
shap.summary_plot(shap_values[:,:-1], X_test_subset, max_display=1000)
```



[54]:
```
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()
```



11.1.4 Performance for cows with 150 days of daily history before calving compared to less than that

[55]:
```
test_set = X_test.merge(y_test, how='inner', left_index=True, right_index=True)
test_set['pred']=y_pred
test_set['pred_prob']=y_pred_prob
relative_features = ['Ydelse_total','robK_BW', 'MILKFAT', 'MILKPROTEIN', 'fpf']
missing_values_cols = [name + '_150_to_120_days_bf_klv' for name in␣
 ↪relative_features]
test_set['missing'] = test_set[missing_values_cols].isnull().any(axis=1).values
```

[56]:
```
missing_compare = test_set.groupby('missing').apply(measures).reset_index()
pp(missing_compare.sort_values(by='PR_AUC',ascending=False))
```

	missing	no_obs	no_ketose	PR_AUC	ROC_AUC	precision	recall	f1
0	False	724.0	92.0	0.5924	0.885766	0.331897	0.836957	0.475309
1	True	165.0	14.0	0.3239	0.779092	0.200000	0.642857	0.305085

As can be seen, performance is again much greater for cows with 150 days of daily history before calving.

12 Trade-off between precision and recall

As can be seen in the precision-recall curve, there is a trade-off between precision and recall. Depending on what is more important for the farmer, different thresholds for the model can be chosen.
This section only looks at the data with 150 days of daily history before calving.

[57]:
```
```

```
test_subset = test_set[test_set['missing']==False]
precision, recall, thresholds = precision_recall_curve(test_subset['target'],␣
↪test_subset['pred_prob'])
thresholds_new = np.concatenate((thresholds, [0]))
find_thresholds = pd.DataFrame({'threshold': thresholds_new, 'precision':␣
↪precision, 'recall': recall})
```

If precision is more important than recall

[58]:
```
threshold = find_thresholds.loc[find_thresholds[find_thresholds['precision']>=0.
↪79]['recall'].idxmax()]['threshold']
pp(classification_report(y_true=test_subset['target'].
↪values,y_pred=test_subset['pred_prob'] >= threshold))
```

```
('              precision    recall  f1-score   support\n'
 '\n'
 '       False       0.90      0.99      0.94       632\n'
 '        True       0.79      0.21      0.33        92\n'
 '\n'
 '    accuracy                           0.89       724\n'
 '   macro avg       0.84      0.60      0.63       724\n'
 'weighted avg       0.88      0.89      0.86       724\n')
```

If recall is more important than precision

[59]:
```
threshold = find_thresholds.loc[find_thresholds[find_thresholds['recall']>=0.
↪94]['precision'].idxmax()]['threshold']
pp(classification_report(y_true=test_subset['target'].
↪values,y_pred=test_subset['pred_prob'] >= threshold))
```

```
('              precision    recall  f1-score   support\n'
 '\n'
 '       False       0.99      0.64      0.78       632\n'
 '        True       0.28      0.95      0.43        92\n'
 '\n'
 '    accuracy                           0.68       724\n'
 '   macro avg       0.63      0.79      0.60       724\n'
 'weighted avg       0.90      0.68      0.73       724\n')
```

Or if both are important

[60]:
```
threshold = find_thresholds.loc[find_thresholds[find_thresholds['recall']>=0.
↪73]['precision'].idxmax()]['threshold']
pp(classification_report(y_true=test_subset['target'].
↪values,y_pred=test_subset['pred_prob'] >= threshold))
```

```
('              precision    recall  f1-score   support\n'
 '\n'
 '       False       0.96      0.88      0.92       632\n'
```

```
'        True       0.47      0.74      0.57        92\n'
'\n'
'    accuracy                           0.86       724\n'
'   macro avg       0.71      0.81      0.75       724\n'
'weighted avg       0.90      0.86      0.87       724\n')
```

Through training on only ‘similar’ (the best performing) ‘bedrifter’ and through selecting the most important features, performance is greatly improved.