blatt09_Guth_Venker_Jaekel

June 29, 2021

1 Abgabe SMD Blatt 09

1.0.1 von Nico Guth, David Venker, Jan Jäkel

[1]:
```
class PDF(object):
    '''
    Displays PDF files in Jupyter Notebooks
    '''
    def __init__(self, pdf, size=(200,200)):
        self.pdf = pdf
        self.size = size

    def _repr_html_(self):
        return '<iframe src={0} width={1[0]} height={1[1]}></iframe>'.
↪format(self.pdf, self.size)

    def _repr_latex_(self):
        return r'\includegraphics[width=1.0\textwidth]{{{0}}}'.format(self.pdf)
```

2 Aufgabe 18 Projektaufgabe: Feature Engineering

2.1 a) Optimieren Sie den „roc auc score“, indem Sie neue Features generieren.

Der Code ist weiter unten im PDF eingebunden.

2.2 b) Wie hoch ist der größte „roc auc score“ den Sie erreichen können?

roc auc score = 0.870

2.3 c) Was würde ein „roc auc score“ von 0.5 bedeuten, was ein „roc auc score“ von 1?

- 0.5 : Die Klassifizierung ist genau so gut wie zufälliges Raten
- 1.0 : Die Klassifizierung ist perfekt auf den gegebenen Daten -> Overfitting

2.4 d) Wie ändern sich Genauigkeit und Sensitivität mit steigendem „prediction threshold“?

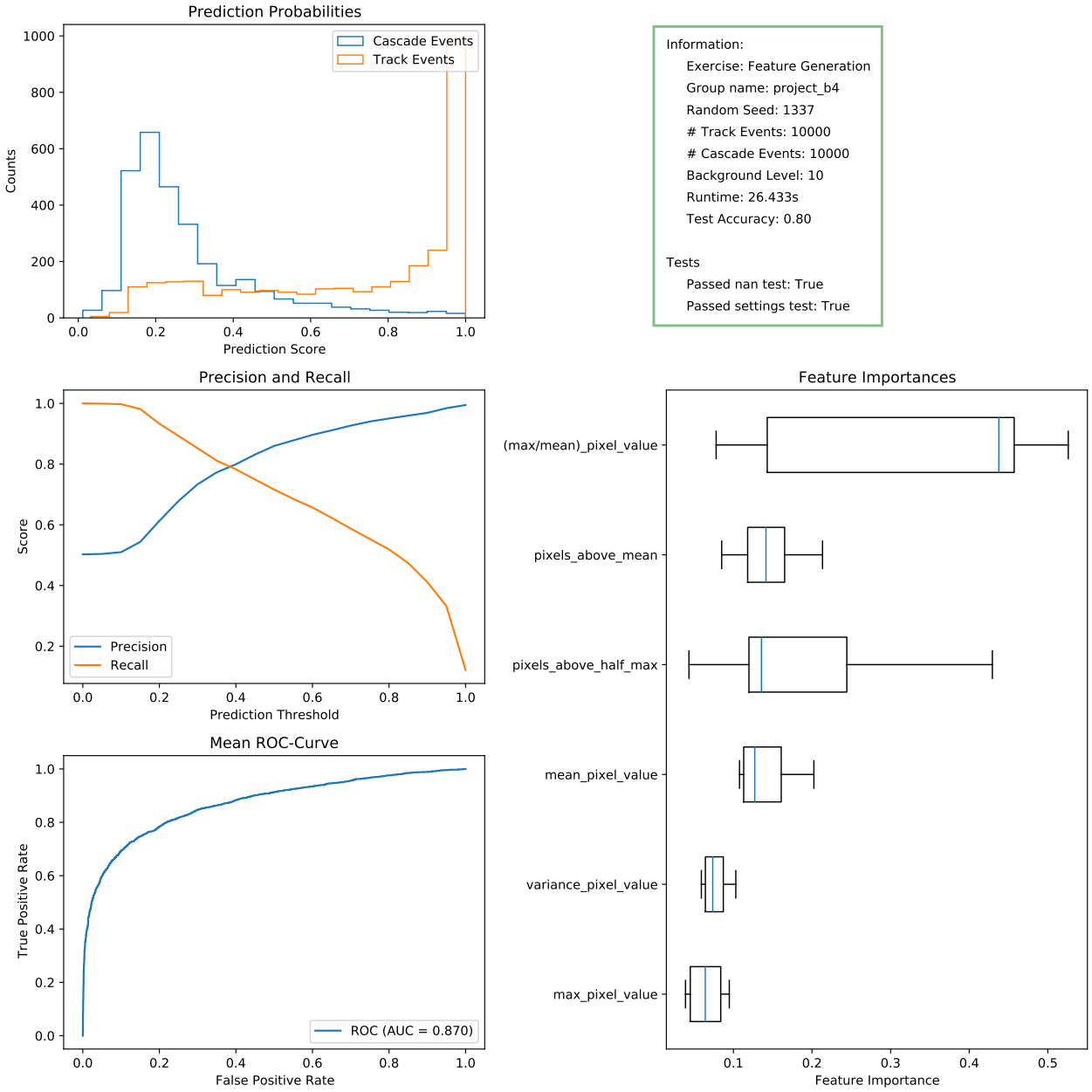
- Bei hohem Prediction Threshold steigt der Recall bzw. Sensitivität (TP/TP+FN)

- Bei niedrigem Prediction Threshold steigt die Precision bzw. Genauigkeit (TP/TP+FP)

2.5 e) Übersichts PDF

[2]:
```
PDF('feature_generation.pdf', size=(500,500))
```

[2]:



3 Aufgabe 19 Projektaufgabe: Energieregression

3.1 a) Implementieren Sie den Random Forest Regressor b) Implementieren Sie 5-Fache Kreuzvalidierung

Der Code ist weiter unten im PDF eingebunden.

3.2 c) Regressor Confusion

3.2.1 Beschreiben Sie kurz, was eine Migrationsmatrix ist.

Wenn eine Migrationsmatrix eine Konfusionsmatrix ist:
In der Konfusionsmatrix kann man ablesen für welchen wahren Wert x wie viele Ereignisse als y vorhergesagt wurden.
Hier kann man also die Zusammenhänge der wahren Werte zu den vorhergesagten Werten einschätzen und so ein Modell bewerten.

3.2.2 Welche Eigenschaften Ihres Regressors können Sie von der Konfusionsmatrix ablesen?

Die Konfusionsmatrix ist auf der Diagonalen am größten und ungefähr um die Diagonale normalverteilt.
Also kann das Modell die Energie recht gut vorhersagen.
Aufgrund der log-log-Skala sollten die Abstände nach oben rechts kleiner werden. Allerdings ist oben rechts die Streuung recht groß. Also sieht man außerdem, dass große Energien nicht so gut vorhergesagt werden. Kleine Energien werden auch teilweise über eine Größenordung falsch vorhergesagt.

3.3 d) Welche Eigenschaften Ihres Regressors können Sie an den Parametern „Bias“ und „Resolution“ ableiten?

Annahme: Mit Resolution ist Variance gemeint.

Bei Energien um 10^4 bis 10^5 GeV ist der Bias und die Variance niedrig.
Bei kleineren und größeren Energien wird sowohl Bias als auch Variance höher.

Also wird die Energie dort ungenauer aufgelöst (Variance) und im mittel zu hoch bzw. zu niedrig eingeschätzt. (Bias)
Niedrige Energien werden eher zu hoch eingeschätzt und hohe Energien eher zu niedrig.
Am besten scheint der Regressor zwischen $2 \cdot 10^4$ und $2 \cdot 10^5$ GeV zu funktionieren.

3.4 e) Welche drei Features haben für die Energieschätzung in diesem Beispiel die größte Bedeutung?

Mit Abstand hat die Anzahl der Pixel mit einem Wert über dem Mittelwert die höchste Bedeutung. Gefolgt davon sind die Anzahl der Pixel mit einem Wert über der hälfte des Maximums und der maximale Wert durch den Mittelwert.

In etwas anderer Reihenfolge sieht man auch in Aufgabe 18, dass diese Features am aussagekräftigsten sind.

3.5 f) Kann es hilfreich sein die zu schätzende Zielgröße (Teilchenenergie) zu skalieren?

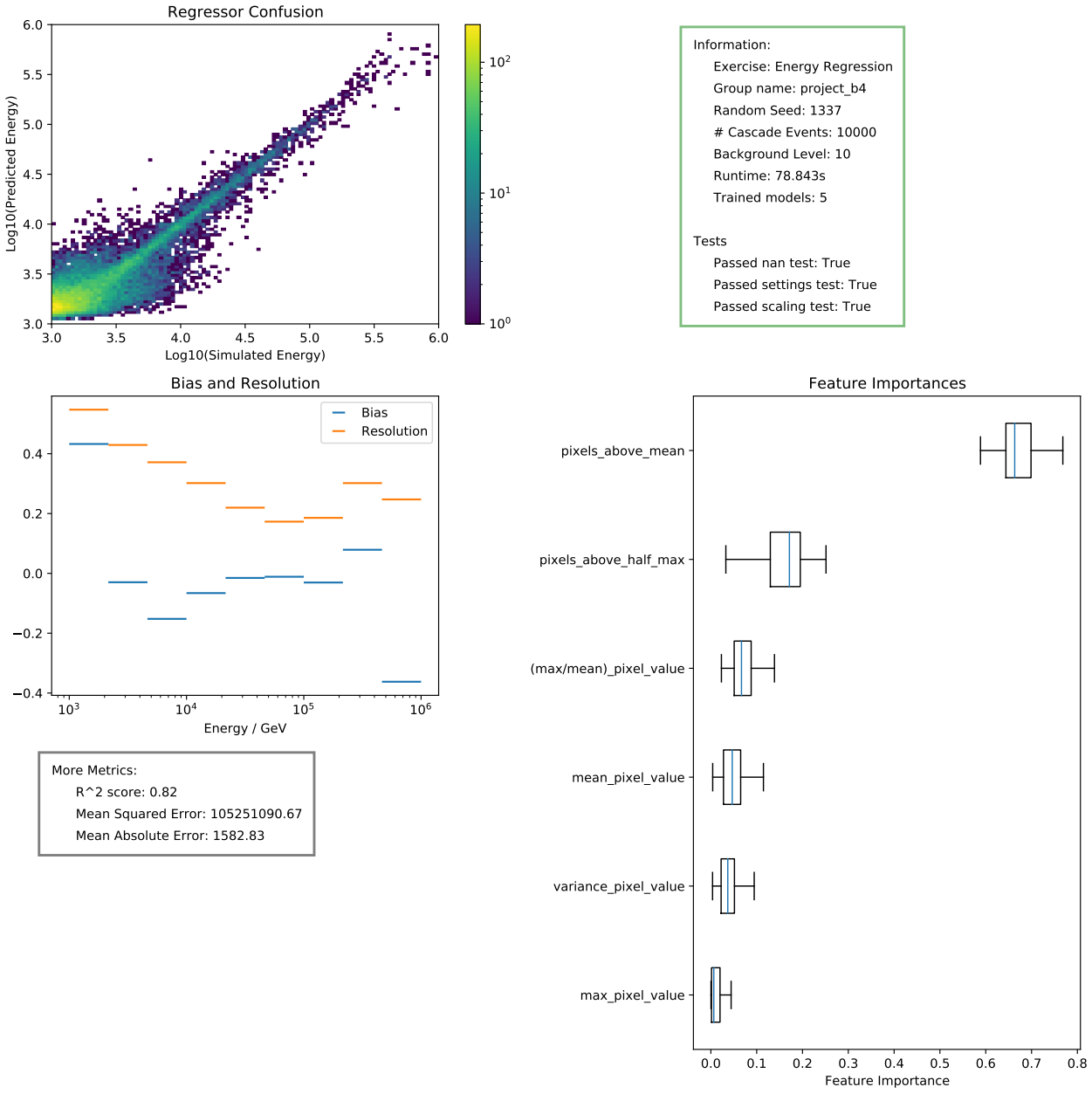
Ja, denn hohe Werte können den Regressor teilweise daran hindern niedrige Werte gut einschätzen zu können.
Z.B. wenn man als Loss-Function den Mean-Squared-Error benutzt, ist es ohne Skalierung deutlich

günstiger hohe Teilchenenergien richtig einzuschätzen als niedrige, obwohl die Problemstellung unter Umständen mehr an niedrigen Werten interessiert sein könnte.

3.6 g) Übersichts PDF

[3]:
```
PDF('energy_regression.pdf', size=(500,500))
```

[3]:



[ ]:

project_b4/reconstruction/preprocessing/feature_generation.py
2021-06-25T15:49+02:00

```python
import numpy as np
import matplotlib.pyplot as plt

class FeatureGenerator():
    """This class generates features for an event

    Based on the values measured in the detector, arbitrary features
    can be generated. These can be used for machine learning exercises
    such as classifying the particle type or estimating energy and origin.
    """

    def __init__(self, detector):
        """Sets up the feature generator

        Parameters
        ----------
        detector : Detector object
            A detector object for which the feature generator will be set up.
        """
        self.detector = detector

    def analyse(self, event):
        """Reconstruct an event measured in the assigned detector.

        Parameters
        ----------
        event : Event
            The event which we want to reconstruct.

        Returns
        -------
        Features: Dict
            A dictionary of arbitrary features.
            Values are to be scalar to be fed into a random forest
        """
        # ---------
        # Exercise 18 (Sheet 9):
        # ---------
```

```
        # Feature Generation (exercises/feature_generation.py)

        mean = np.mean(event.pixels)
        max = np.max(event.pixels)

        features = {
            'max_pixel_value': max,
            'variance_pixel_value': np.var(event.pixels),
            'mean_pixel_value': mean,
            '(max/mean)_pixel_value': max/mean,
            'pixels_above_half_max': (event.pixels > 0.5*max).sum(),
            'pixels_above_mean': (event.pixels > mean).sum()
        }


        return features
```

project_b4/reconstruction/machine_learning/energy_regression.py
2021-06-25T18:25+02:00

```python
import numpy as np
from sklearn import (
    ensemble, linear_model, neighbors, svm, tree, naive_bayes,
    gaussian_process, neural_network, dummy)
from sklearn.model_selection import KFold
from sklearn.base import clone
from tqdm import tqdm


def define_model(seed):
    """A helper function to retrieve a model for the energy regression
    exercise. The parameter seed gets used to set the random state of
    the model
    and ensure reproducible results"""

    # ---------
    # Exercise 19 (Sheet 9):
    # ---------
    # Energy_regression (exercises/energy_regression.py)

    model = ensemble.RandomForestRegressor(random_state=seed)

    return model


def cross_validate_model(X, y, model, seed):
    """This function implements a cross validation on a given model.

    Required return values:
    -----------------------
    predictions: np.array of the same shape as y
        (These are the predictions on the test sets combined into one
    array)
    true_values: np.array of the same shape as y
        (These are the y's chosen in the different cross validation
    steps
        combined into one array. This equals y but with different
    order.)
    models: list of (n_cross_validation_steps) models
        These are used to calculate the feature importances later
```

```
    """
    # ---------
    # Exercise 19 (Sheet 9):
    # ---------
    # Energy_regression (exercises/energy_regression.py)

    kf = KFold(n_splits=5, shuffle=True, random_state=seed)

    predictions = []
    true_values = []
    models = []

    for train_index, test_index in kf.split(X):
        X_train, y_train = X.iloc[train_index], y[train_index]
        X_test, y_test = X.iloc[test_index], y[test_index]

        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

        predictions.append(y_pred)
        true_values.append(y_test)
        models.append(model)

    predictions = np.concatenate(predictions)
    true_values = np.concatenate(true_values)

    return predictions, true_values, models
```