STØTTET AF
Promilleafgiftsfonden for landbrug

Legume proportion in clover grass

Contact info: SEGES-datascience@seges.dk

Data Acquisition and Preprocessing

In this project, we utilize a dataset derived from NIRS sensors, organized into different groups based on their acquisition time and source characteristics. We also make use of data from the ATV-experiment by Aarhus University.:

- **ATV-data Collection**: Includes shapefiles from various geographical locations collected over several years, starting from 2018.
- **NIR-2022-data Series**: Consists of Excel files with data for the entire year of 2022.
- **NIR-2023 Monthly Data Sets**: Monthly data for 2023, segmented into separate collections for each month, available in Excel format.

Each dataset has undergone comprehensive preprocessing, including data cleaning, normalization, and aggregation, to ensure data quality and consistency for analysis.

Data Processing and Analysis Workflow

Data Loading and Merging:

Each group of raw data files was initially loaded into our cleaning environment. The data from individual files within each group were then merged to form datasets for each respective data collection.

Data Cleaning and Quality Assurance:

In the subsequent phase, we conducted a thorough cleaning process. This involved:

- Eliminating observations from areas outside the designated fields.
- Identifying and marking fields exhibiting unreliable characteristics. These included fields not fully covered, those showing atypical development in clover proportion throughout the season, and fields where clover proportions deviated from the expected 0-100% range.

Rasterization for Spatial Analysis:

The cleaned and refined data was then transformed into a raster format. This process entailed the creation of a grid structure with 10x10 meter intervals. Each grid cell in this structure represents a specific 10x10 meter area within a field, enabling a spatial analysis of the data.

Data structure

Target

The target value is a floating-point number ranging from 0 to 100 representing the clover-proportion for a 10x10 meter grid on a given field.

Features

As input to the model we make use of several different features. These features are categorized into distinct groups based on their nature and source.

Temporal features

- **Time since January (time_since_jan)**: This feature captures the number of days elapsed since January 1st, providing a temporal context for each clover proportion registration.

Satellite data

We utilize satellite imagery from Sentinel 2, specifically the Level 2A product which includes atmospheric correction. The imagery encompasses 11 distinct spectral bands, each capturing different aspects of the earth's surface:

- **B1 (Coastal Aerosol Band)**
- **B2 (Blue Band)**
- **B3 (Green Band)**
- **B4 (Red Band)**
- **B5 (Vegetation Red Edge Band)**
- **B6 (Vegetation Red Edge Band)**
- **B7 (Vegetation Red Edge Band)**
- **B8 (NIR Band)**
- **B8A (Narrow NIR Band)**
- **B9 (Water Vapor Band)**
- **B11 (SWIR Band)**
- **B12 (SWIR Band)**

Additionally, we derive several vegetation indices from these raw bands to further enhance our analysis:

- **NDVI (Normalized Difference Vegetation Index)**: Indicates vegetation health and density.
- **NDRE (Normalized Difference Red Edge Index)**: Useful for assessing crop health and stress.
- **gNDVI (Green NDVI)**: A variant of NDVI using the green band, effective in dense vegetation areas.
- **GCI (Green Chlorophyll Index)**: Provides an estimation of chlorophyll concentration in plants.

Terrain data

- **Field height mean ('field_height_mean')**: This feature represents the average height of the field above the surrounding terrain, measured in meters. It is crucial for understanding the topographical variations within the field, which can impact plant growth and health.

Each feature-value are collected for each 10x10 meter grid on the respective field.

Models

The following model is used:

- Gradient tree boosting model - xgboost

Evaluation structure and metrics

- The model was trained using a tweedie loss-function and evaluated using the mean absolute error (mae).
- The model was trained on the 10x10 meter grid observations but aggregated to field-level for evaluation. Thus, the final mae provided denotes the performance on aggregated data (after training).

Experiment

- Initially, fields identified with unreliable characteristics during the data cleaning phase were excluded from the analysis.
- Observational data collected between June 15 and September 1 were eliminated. This period, defined as the summer season, typically exhibits excessive noise in satellite imagery due to the presence of vegetation other than clover, thereby impacting the model's ability to accurately estimate clover proportions.
- To achieve a balanced representation across the entire 0 to 100% range of clover proportions, the dataset was stratified into bins. Each bin spanned a 10% range, with a uniform sampling of 20,000 data points. This stratification approach was validated, as varying the sample size per bin showed no significant impact on the results.
- Various model configurations were tested, involving different combinations of features and hyperparameters.
- The models' performances were evaluated using a K-fold cross-validation approach. Here, 'k' represented the data from a unique farmer's fields. For each fold, the model was trained on data excluding one farmer's fields, and then tested on that excluded set. This process ensured that each farmer's data was used exactly once as the validation set. The Mean Absolute Error (MAE) was computed for each farmer, and the average MAE across all farmers was used to assess the overall model performance.
- To prevent overfitting, each model incorporated early stopping based on in-sample data performance.
- The model with the lowest MAE score in the cross-validation process was selected as the best performing model. Based on the lowest score on the evaluation-fields this model

was most suitable for estimating clover proportions across different fields not used during training.
- To assess the most important features, we derive the 'gain' from each fold and average the 'gain' for each feature across all folds. This illustrate what features on average is most important for the final performance.

Conclusion

Based on the results from the above experiment we can make the following conclusions:

- A large amount of the data are removed (almost 50%) based on unreliable patterns. This suggests that the target variable is associated with a large degree of uncertainty.
- Almost all the training data are observed in September 2023 making the models ability to generalize highly dependent on the September 2023 traits.
- The most important features are days since january 01 (time_since_jan) and the field height (field_height_mean) which signifies that the S2-bands are insufficient to predict the clover-proportion without other features as well. This is true, at least on a 10m resolution.
- The model performs almost twice as good as a simple mean-prediction-model.

Future work

The following areas are interesting for future work:

Model

Integration of Computer Vision Techniques: Implementing sophisticated Convolutional Neural Network (CNN) architectures to process high-resolution satellite imagery. This approach aims to leverage increased spatial detail for more accurate analysis.

Incorporating Time-Series Analysis: Investigating the impact of adding a temporal dimension to track clover proportion changes throughout the season. This will assess whether considering time-series data can significantly enhance model accuracy.

Data

Requirement for High-Resolution Satellite Data: Exploring the necessity of satellite data with a resolution finer than 10 meters.

Assessment of NIRS Sensor Accuracy: Conducting a thorough evaluation of the NIRS sensor's validity. This includes comparing model performance using NIRS-derived data against data from alternative sources, such as those obtained from Perkins.

Impact of Cloud Cover on Model Performance: Analyzing how the likelihood of cloud cover influences the model's effectiveness. Specifically focusing on understanding the correlation between increased cloud probability and potential degradation in model accuracy.

Technical setup

In this notebook the py39_pla_clover conda environment is used. It can be installed using the environment.yml file located in the GitHub repository.

Implementation and Analysis

In the following we show the final analysis and overview of the data used.

Notebook Configuration and Initial Setup

In [ ]:
```
import warnings
import itertools
import pprint
import numpy as np
import pandas as pd
import geopandas as gpd
from pathlib import Path
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn
import xgboost as xgb
from collections import defaultdict
```

In [ ]:
```
# Set figure size for the notebook
plt.rcParams["figure.figsize"]=(10,6)
plt.rcParams['figure.dpi']=100

# set seaborn whitegrid theme
sns.set(style="whitegrid")
```

Exploratory Data Analysis (EDA)

In [ ]:
```
data_folder = Path('data/final_datasets/') #local data folder
```

In [ ]:
```
df_training_data = pd.read_parquet(data_folder / 'dataset_full.parquet')
```

In [ ]:
```
print('The training dataset before filtering consists of {} field-ids across {} farms
     df_training_data.reset_index()['field_idx'].nunique(), df_training_data['farm']
    )
)
```

```
The training dataset before filtering consists of 1154 field-ids across 56 farms
```

In [ ]:
```
df_training_data_filtered = df_training_data[
                            ~ df_training_data['negative_values'] &
                            ~ df_training_data['farm_negative_values'] &
                            ~ df_training_data['large_values'] &
                            ~ df_training_data['farm_large_values'] &
                            ~ df_training_data['unreliable'] &
```

```
                            ~ df_training_data['broad_unreliable'] &
                            ~ df_training_data['negative_develop_clover'] &
                            ~ df_training_data['not_covered'] &
                            ~ df_training_data['weird']
                          ]
```

In [ ]:
```
print('After filtering out unreliable fields, the dataset consists of {} field-ids a
    df_training_data_filtered.reset_index()['field_idx'].nunique(), df_training_data
    )
)
```

```
After filtering out unreliable fields, the dataset consists of 599 field-ids across 5
0 farms
```

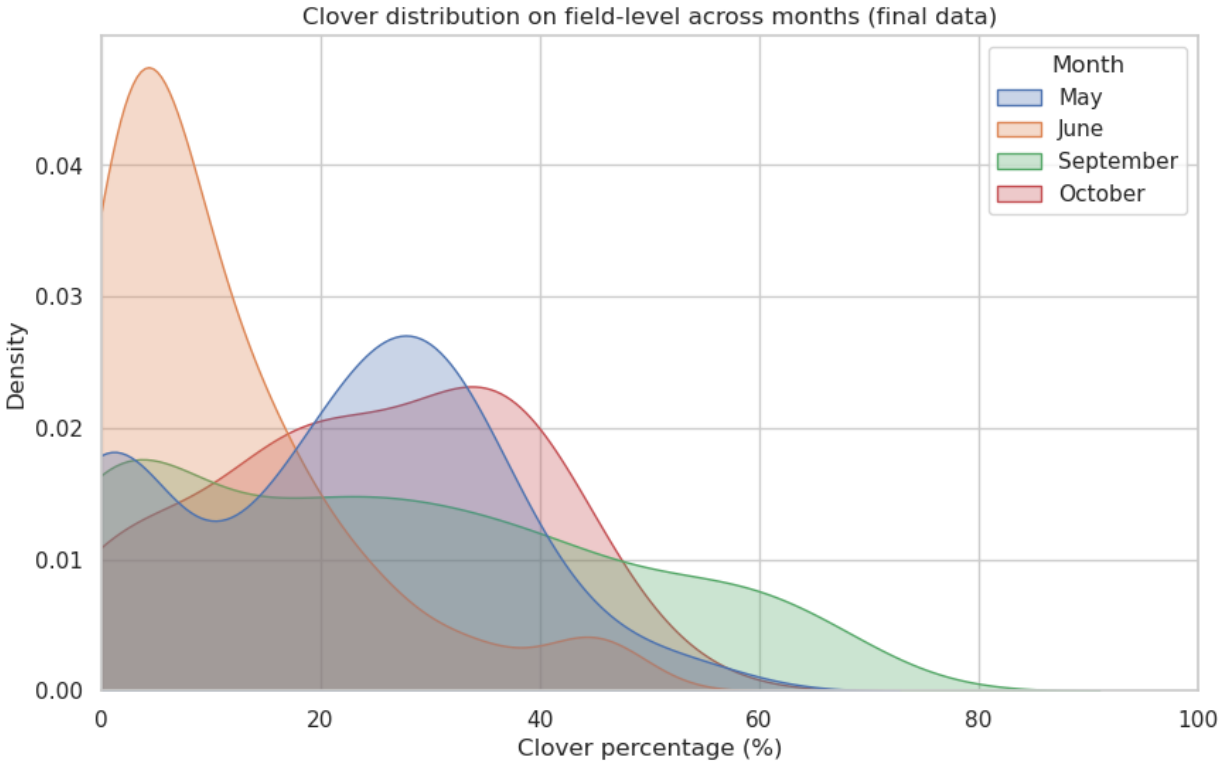
In [ ]:
```
df_training_data_filtered = df_training_data_filtered[
    (df_training_data_filtered['time_since_jan'] <= 165)
  | ((df_training_data_filtered['time_since_jan'] >= 243))]
```

In [ ]:
```
print('After filtering out data from the summer months, the dataset consists of {} f
    df_training_data_filtered.reset_index()['field_idx'].nunique(), df_training_data
    )
)
```

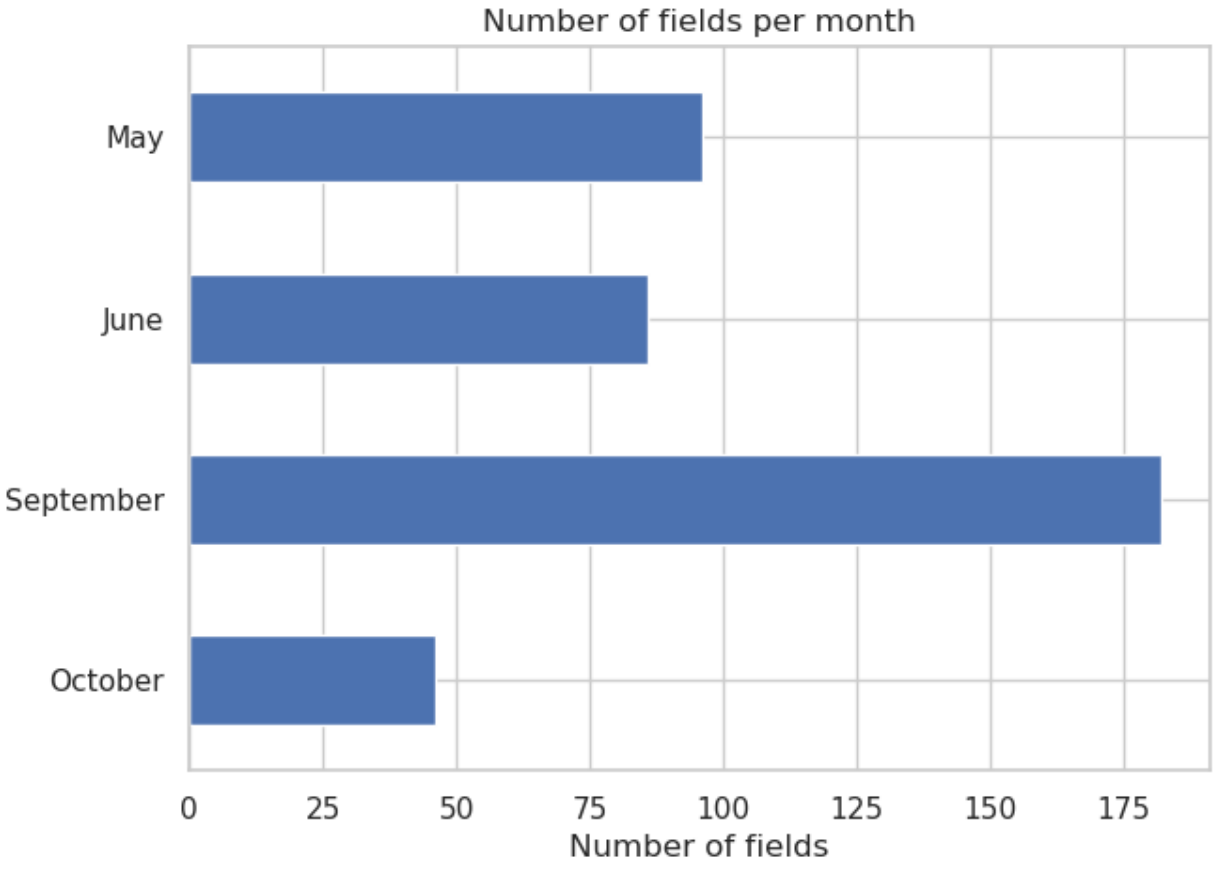
```
After filtering out data from the summer months, the dataset consists of 410 field-id
s across 44 farms
```

In [ ]:
```
# Copying and resetting index of the training data
df_plot_across_months = df_training_data_filtered.copy().reset_index()

# Grouping by 'field_idx' and aggregating relevant statistics
df_plot_across_months = df_plot_across_months.groupby('field_idx').agg({
    'clover_per': 'mean',
    'time': 'first',
    'dataset': 'first',
    'field': 'count'
})

# Mapping and ordering the months
month_mapping = {'05': 'May', '06': 'June', '09': 'September', '10': 'October'}
month_order = ['May', 'June', 'September', 'October']
df_plot_across_months['Month'] = df_plot_across_months['time'].dt.strftime('%m').map

# Preparing the data for the field count plot
df_field_count_across_months = df_plot_across_months.reset_index().groupby('Month')[
df_field_count_across_months.index = pd.Categorical(df_field_count_across_months.ind
                                                    categories=month_order,
                                                    ordered=True)
df_field_count_across_months.sort_index(ascending=False, inplace=True)
```

In [ ]:
```
sns.kdeplot(data=df_plot_across_months, x='clover_per', hue='Month', fill=True,
            common_norm=False, alpha=0.3, bw_adjust=1, hue_order=month_order)
plt.title('Clover distribution on field-level across months (final data)')
plt.xlabel('Clover percentage (%)')
_ = plt.xlim(0,100)
```



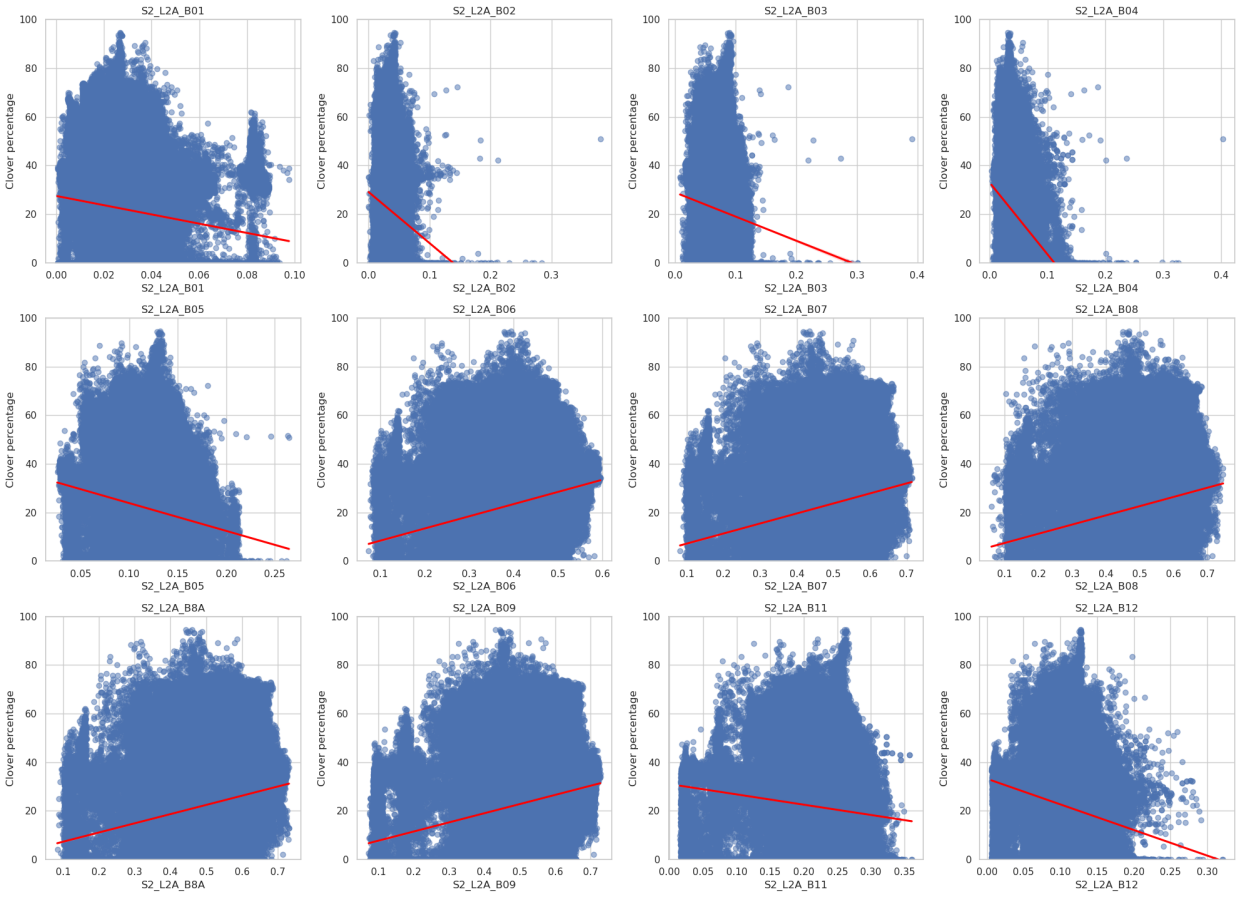
In [ ]:
```
df_field_count_across_months.plot(kind='barh', figsize=(7,5))
plt.title('Number of fields per month')
_ = plt.xlabel('Number of fields')
```



In [ ]:
```
y_var = 'clover_per'
x_vars = ['S2_L2A_B01', 'S2_L2A_B02', 'S2_L2A_B03', 'S2_L2A_B04',
          'S2_L2A_B05', 'S2_L2A_B06', 'S2_L2A_B07', 'S2_L2A_B08',
          'S2_L2A_B8A', 'S2_L2A_B09', 'S2_L2A_B11', 'S2_L2A_B12']
```

```
fig, axes = plt.subplots(3, 4, figsize=(20, 15))
axes = axes.flatten()

for i, x_var in enumerate(x_vars):
    sns.regplot(data=df_training_data_filtered, x=x_var, y=y_var, ax=axes[i],
                scatter_kws={'alpha':0.5}, line_kws={'color':'red'})
    axes[i].set_ylim(0, 100)
    axes[i].set_title(x_var)
    axes[i].set_xlabel(x_var)
    axes[i].set_ylabel("Clover percentage")

fig.suptitle('Relationships between Features and Clover Percentage on grid-level',
             fontsize=25, y=1.02)

plt.tight_layout()
plt.show()
```

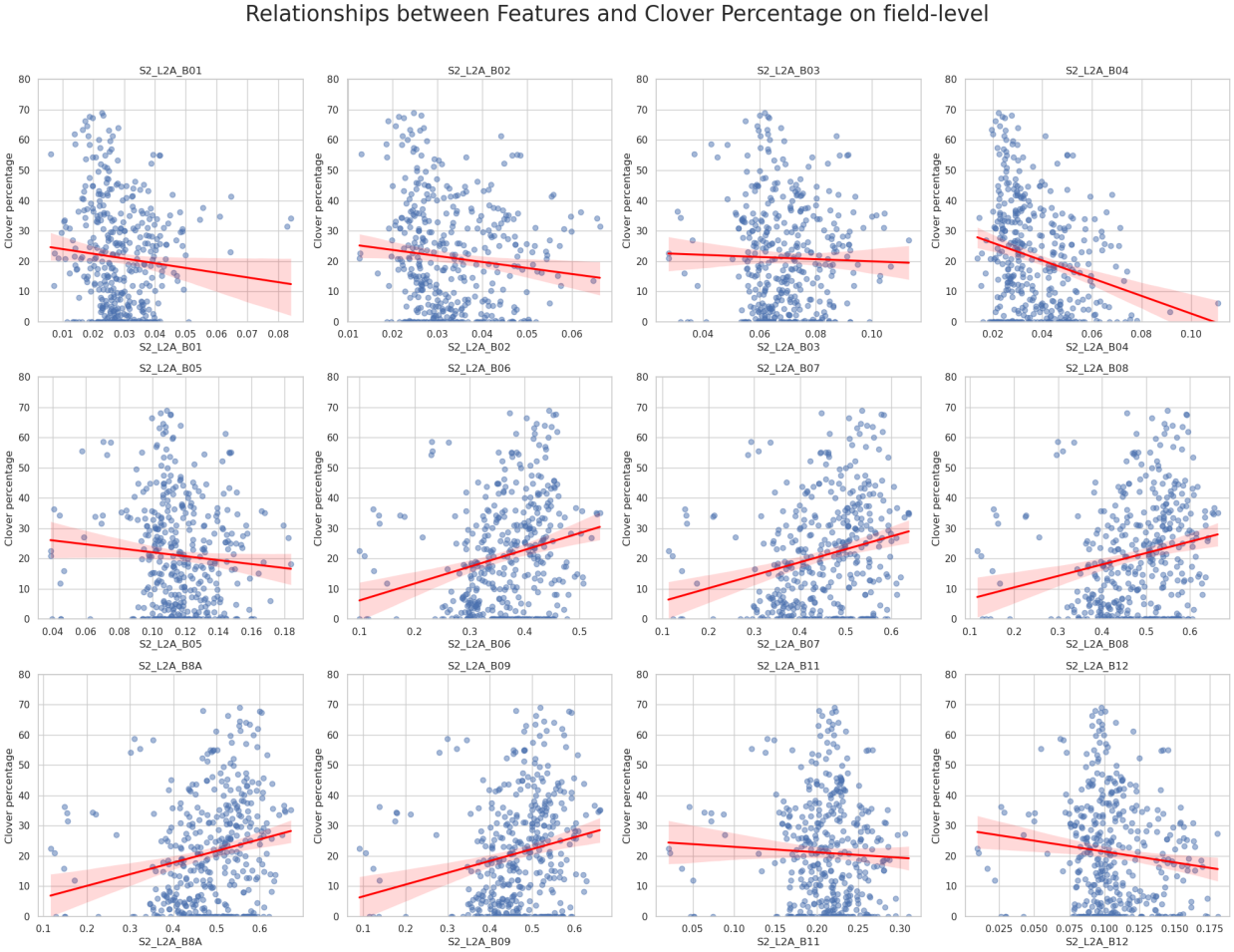
Relationships between Features and Clover Percentage on grid-level



In [ ]:

```
# Aggregate data to field-level
df_data_by_field = df_training_data_filtered.reset_index().copy()
df_data_by_field = df_data_by_field.groupby('field_idx')[[y_var] + x_vars].mean()

fig, axes = plt.subplots(3, 4, figsize=(20, 15))
axes = axes.flatten()

for i, x_var in enumerate(x_vars):
    sns.regplot(data=df_data_by_field, x=x_var, y=y_var, ax=axes[i],
                scatter_kws={'alpha':0.5}, line_kws={'color':'red'})
    axes[i].set_ylim(0, 80)
    axes[i].set_title(x_var)
```

```
    axes[i].set_xlabel(x_var)
    axes[i].set_ylabel("Clover percentage")

fig.suptitle('Relationships between Features and Clover Percentage on field-level',
             fontsize=25, y=1.02)

plt.tight_layout()
plt.show()
```



Feature Engineering

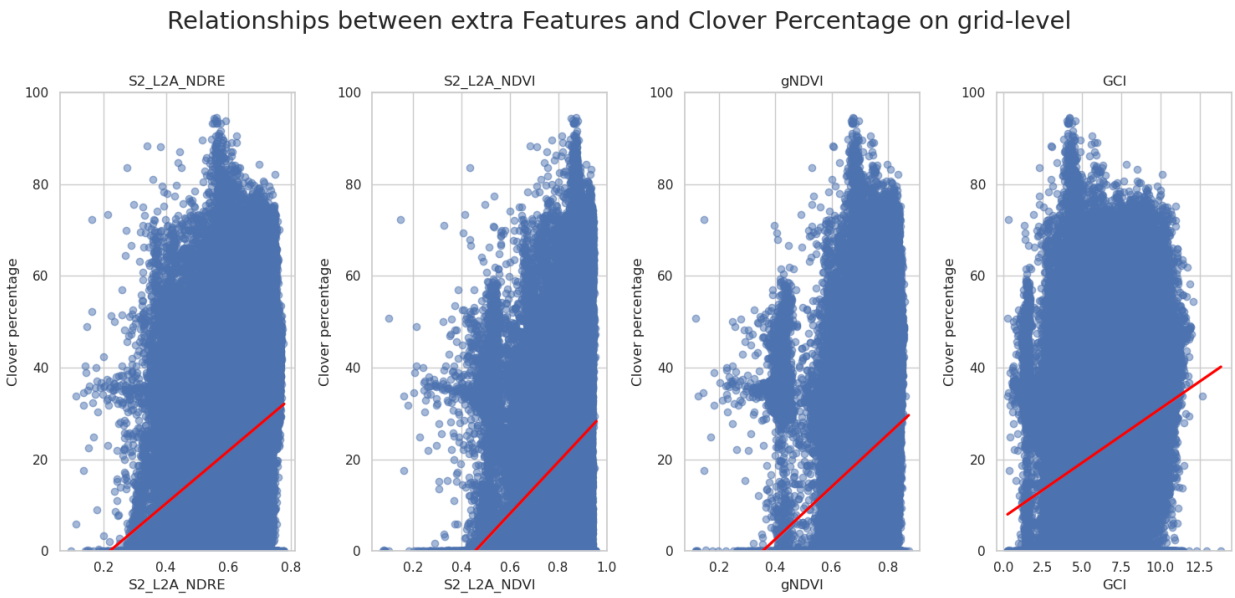
In [ ]:
```
def gNDVI(df, B3, B8):
    gNDVI = (df[B8] - df[B3]) / (df[B8] + df[B3])
    return gNDVI

def GCI(df, B3, B8):
    gci = df[B8] / df[B3] - 1
    return gci
```

In [ ]:
```
df_training_data_filtered['gNDVI'] = gNDVI(df_training_data_filtered, 'S2_L2A_B03',
df_training_data_filtered['GCI'] = GCI(df_training_data_filtered, 'S2_L2A_B03', 'S2_
```

Explore additional features

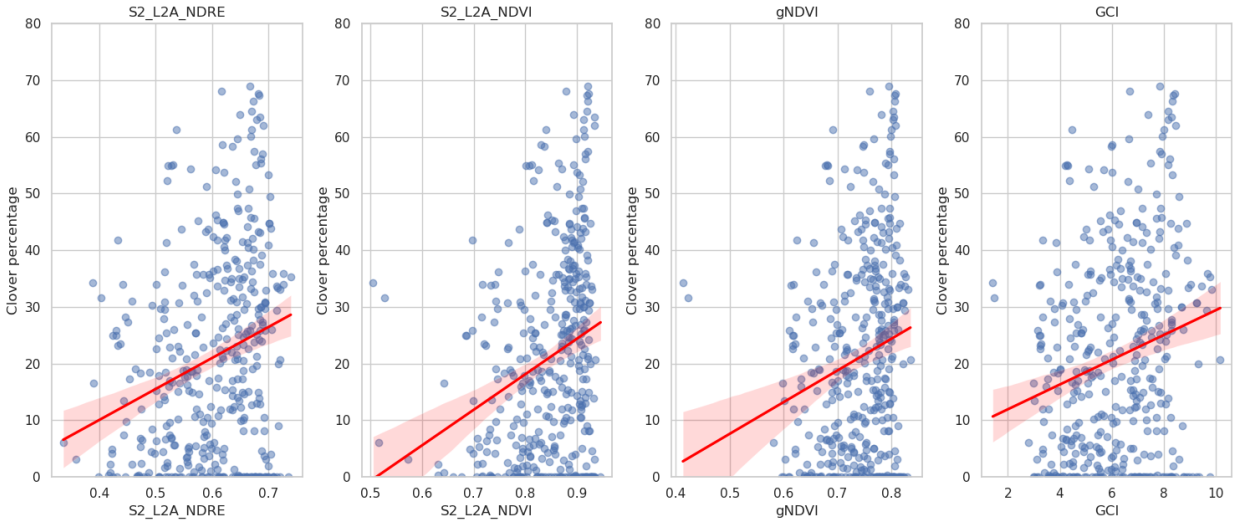
In [ ]:
```
x_vars_new = ['S2_L2A_NDRE', 'S2_L2A_NDVI', 'gNDVI', 'GCI']

fig, axes = plt.subplots(1, 4, figsize=(15, 7))
```

```
axes = axes.flatten()

for i, x_var in enumerate(x_vars_new):
    sns.regplot(data=df_training_data_filtered, x=x_var, y=y_var, ax=axes[i],
                scatter_kws={'alpha':0.5}, line_kws={'color':'red'})
    axes[i].set_title(x_var)
    axes[i].set_xlabel(x_var)
    axes[i].set_ylabel("Clover percentage")
    axes[i].set_ylim(0, 100)

fig.suptitle('Relationships between extra Features and Clover Percentage on grid-leve
             fontsize=21, y=1.02)

plt.tight_layout()
plt.show()
```



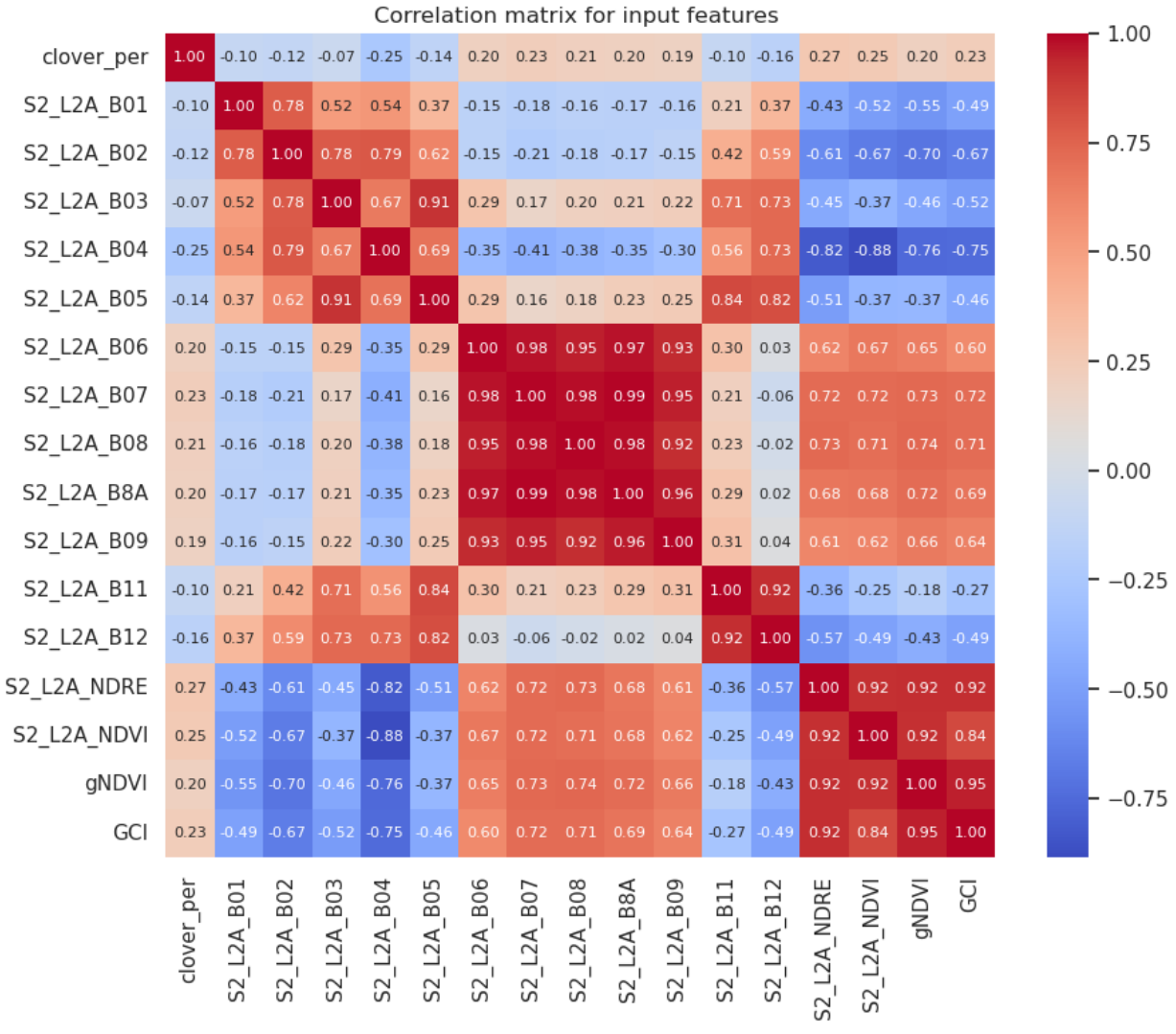
In [ ]:

```
# Aggregate data to field-level
df_data_by_field = df_training_data_filtered.reset_index().copy()
df_data_by_field = df_data_by_field.groupby('field_idx')[[y_var] + x_vars_new].mean(

fig, axes = plt.subplots(1, 4, figsize=(15, 7))
axes = axes.flatten()

for i, x_var in enumerate(x_vars_new):
    sns.regplot(data=df_data_by_field, x=x_var, y=y_var, ax=axes[i],
                scatter_kws={'alpha':0.5}, line_kws={'color':'red'})
    axes[i].set_ylim(0, 80)
    axes[i].set_title(x_var)
    axes[i].set_xlabel(x_var)
    axes[i].set_ylabel("Clover percentage")

fig.suptitle('Relationships between extra Features and Clover Percentage on field-lev
             fontsize=25, y=1.02)

plt.tight_layout()
plt.show()
```



In [ ]:

```
corr_matrix = df_training_data_filtered[['clover_per'] + x_vars + x_vars_new].corr()

# Create the heatmap using the "coolwarm" palette
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap="coolwarm", cbar=True, annot_kws
plt.title('Correlation matrix for input features ')
plt.show()
```



Benchmark Model

In [ ]:
```
target = 'clover_per'
```

In [ ]:
```
# Run k-fold cross-validation
test_sets = []
feature_sets = []
for farm in df_training_data['farm'].unique():
    # Define train/test dataset
    df_test_data = df_training_data.copy()
    df_test_data = df_test_data[df_test_data['farm'] == farm]
    df_train_data = df_training_data.copy()
    df_train_data = df_train_data[~(df_train_data['farm'] == farm)]
    # Seperate features and target
    X_train = df_train_data.drop([target], axis=1)
    X_test = df_test_data.drop([target], axis=1)
    y_train = df_train_data[target]
    y_test = df_test_data[target]
    # Extract features to include
    X_train = X_train[
        sorted(X_train.columns)]
    X_test = X_test[
        sorted(X_test.columns)]

    column_list = list(X_train.columns)

    test_set = X_test.copy()
    test_set['pred'] = y_train.mean()
    test_set['target'] = y_test
    test_set['farm'] = farm

    test_sets.append(test_set)
    del X_train
    del X_test
    del df_train_data
    del df_test_data

# Concatenate results into dataframe and aggregate to field-level
df_predictions_baseline = pd.concat(test_sets)
df_predictions_baseline = df_predictions_baseline.reset_index().groupby('field_idx')
    'target': 'mean',
    'pred': 'mean',
    'farm': 'first'
})
```

In [ ]:
```
# Dictionary to store evaluation output
eval_results_baseline = {}

# Filter away fields with no clover (0.5 since some fields are only approximate zero
df_predictions_base_filtered = df_predictions_baseline[df_predictions_baseline['targ

#MAE
eval_results_baseline['MAE'] = sklearn.metrics.mean_absolute_error(
    df_predictions_baseline['target'], df_predictions_baseline['pred']
    )
#RMSE
```

```
eval_results_baseline['RMSE'] = np.sqrt(sklearn.metrics.mean_squared_error(
    df_predictions_baseline['target'], df_predictions_baseline['pred']
    ))
#R2
eval_results_baseline['R2'] = sklearn.metrics.r2_score(
    df_predictions_baseline['target'], df_predictions_baseline['pred']
    )
#MAE
eval_results_baseline['MAE (>0)'] = sklearn.metrics.mean_absolute_error(
    df_predictions_base_filtered['target'], df_predictions_base_filtered['pred']
    )
#RMSE
eval_results_baseline['RMSE (>0)'] = np.sqrt(sklearn.metrics.mean_squared_error(
    df_predictions_base_filtered['target'], df_predictions_base_filtered['pred']
    ))
#R2
eval_results_baseline['R2 (>0)'] = sklearn.metrics.r2_score(
    df_predictions_base_filtered['target'], df_predictions_base_filtered['pred']
    )

eval_results_baseline = {key : round(eval_results_baseline[key], 2) for key in eval_
```

In [ ]:
```
# (>) denotes only fields with mean clover proportion above 0
pprint.pprint(eval_results_baseline)
```

```
{'MAE': 14.31,
 'MAE (>0)': 13.4,
 'R2': -0.02,
 'R2 (>0)': -0.06,
 'RMSE': 18.1,
 'RMSE (>0)': 18.09}
```

Machine Learning Model Training

In [ ]:
```
dict_model_parameters = {
    'objective': 'reg:tweedie',
    'tree_method': 'approx',
    'eval_metric': 'mae',
    'tweedie_variance_power': 1.18,
    'early_stopping_rounds': 50,
    'colsample_bytree': 0.55,
    'n_estimators': 3000,
    'learning_rate': 0.1,
    'reg_lambda': 800,
    'reg_alpha': 50,
    'max_depth': 6,
}
```

In [ ]:
```
features_to_include = [
    'S2_L2A_B01', 'S2_L2A_B02', 'S2_L2A_B03', 'S2_L2A_B04', 'S2_L2A_B05', 'S2_L2A_B0
    'S2_L2A_B07', 'S2_L2A_B08', 'S2_L2A_B8A', 'S2_L2A_B09', 'S2_L2A_B11','S2_L2A_B12
    'S2_L2A_NDVI', 'S2_L2A_NDRE', 'gNDVI', 'GCI', 'time_since_jan', 'field_height_me
]
```

In [ ]:
```
# Exclude irrelevant columns
df_training_data_cv = df_training_data_filtered.copy()
df_training_data_cv = df_training_data_cv[features_to_include + [target, 'farm']]
```

In [ ]:
```
# Stratisfy training data
df_train_data_stratisfy = df_training_data_cv.copy()

# Create 10%-ranged bins
bins = np.arange(0, 110, 10)
df_train_data_stratisfy['clover_per_bin'] = pd.cut(
    df_train_data_stratisfy['clover_per'], bins=bins,
    labels=bins[1:], right=True, include_lowest=True
    )

# Sample observations in each bin
df_train_data_stratisfy['clover_per_bin'] = df_train_data_stratisfy['clover_per_bin'
df_train_data_stratisfy = df_train_data_stratisfy.groupby('clover_per_bin').apply(
    lambda x: x.sample(20000, random_state=123, replace=True)
    )

# Reset index
df_train_data_stratisfy = df_train_data_stratisfy.reset_index(level=0, drop=True)
```

In [ ]:
```
# Run k-fold cross-validation
test_sets = []
feature_sets = []
for farm in df_training_data_cv['farm'].unique():
    # Define train/test dataset
    df_test_data = df_training_data_cv.copy()
    df_test_data = df_test_data[df_test_data['farm'] == farm]
    df_train_data = df_train_data_stratisfy.copy()
    df_train_data = df_train_data[~(df_train_data['farm'] == farm)]
    # Seperate features and target
    X_train = df_train_data.drop([target], axis=1)
    X_test = df_test_data.drop([target], axis=1)
    y_train = df_train_data[target]
    y_test = df_test_data[target]
    # Extract features to include
    X_train = X_train[features_to_include]
    X_test = X_test[features_to_include]
    column_list = list(X_train.columns)

    # Initiate model
    model = xgb.XGBRegressor(**dict_model_parameters)

    # Fit model
    model.fit(
        X_train, y_train,
        eval_set=[(X_train, y_train), (X_test, y_test)],
        verbose=True
    )

    # Force negative predictions to zero
    y_pred = model.predict(X_test)
    y_pred[y_pred < 0] = 0
    test_set = X_test.copy()
    test_set['pred'] = y_pred
    test_set['target'] = y_test
```

```
    test_set['farm'] = farm
    test_set['best_iteration'] = model.best_iteration

    # Append fold result to output list
    test_sets.append(test_set)
    feature_sets.append(model.get_booster().get_score(importance_type='gain'))
    del X_train
    del X_test
    del df_train_data
    del df_test_data

# Concatenate results into dataframe and aggregate to field-level
df_predictions = pd.concat(test_sets)
df_predictions = df_predictions.reset_index().groupby('field_idx').agg({
    'target': 'mean',
    'pred': 'mean',
    'farm': 'first',
    'best_iteration': 'mean'
})
```

Evaluate results

In [ ]:
```
# Merge field-features onto results
features_to_add = ['time','dataset']
df_predictions = df_predictions.merge(
    df_training_data_filtered.reset_index().groupby('field_idx')[features_to_add].fi
    left_index=True, right_index=True, how='inner')
```

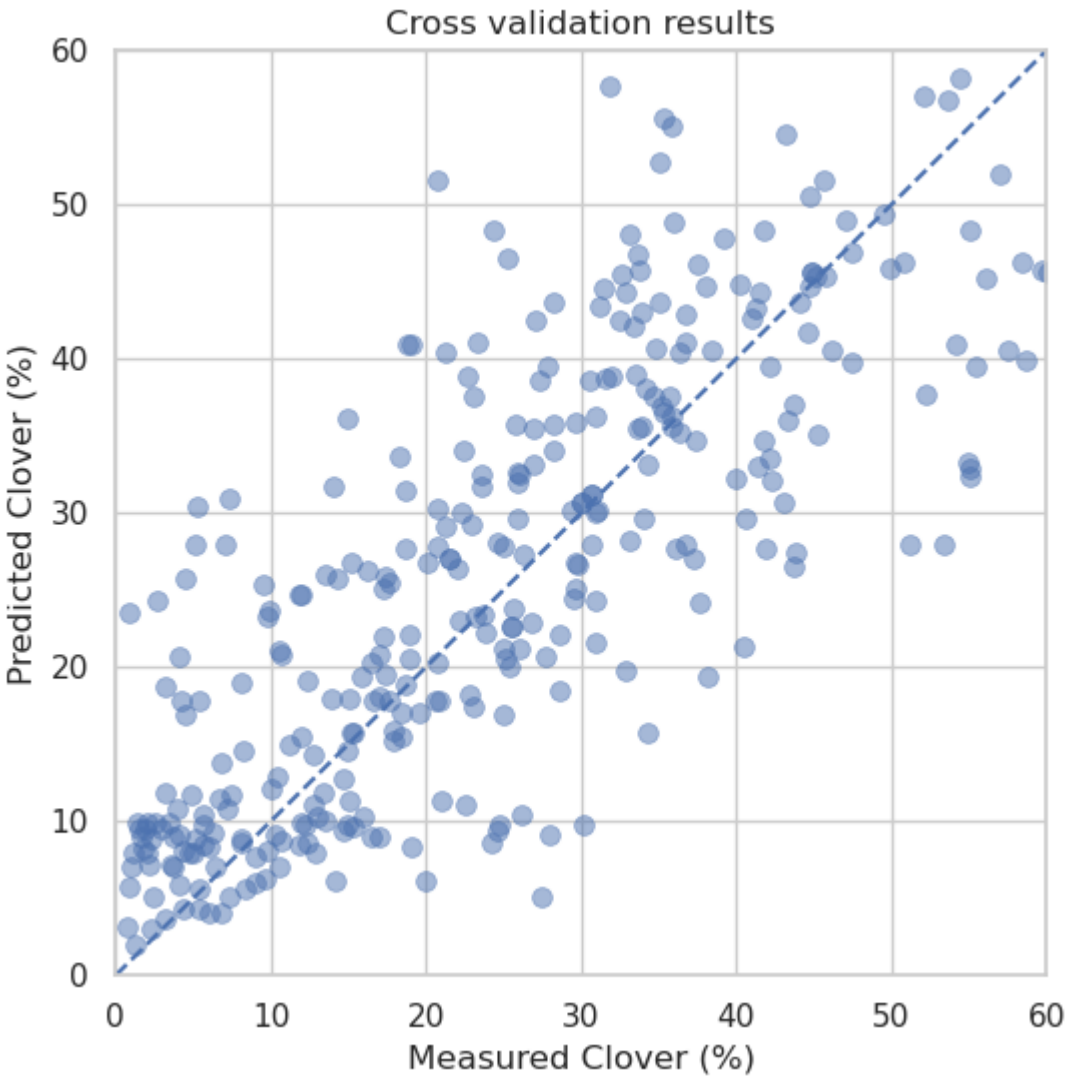
In [ ]:
```
# Dictionary to store evaluation output
eval_results = {}

# Filter away fields with no clover (0.5 since some fields are only approximate zero)
df_predictions_filtered = df_predictions[df_predictions['target']>0.5]

#MAE
eval_results['MAE'] = sklearn.metrics.mean_absolute_error(
    df_predictions['target'], df_predictions['pred']
    )
#RMSE
eval_results['RMSE'] = np.sqrt(sklearn.metrics.mean_squared_error(
    df_predictions['target'], df_predictions['pred']
    ))
#R2
eval_results['R2'] = sklearn.metrics.r2_score(
    df_predictions['target'], df_predictions['pred']
    )
#MAE
eval_results['MAE (>0)'] = sklearn.metrics.mean_absolute_error(
    df_predictions_filtered['target'], df_predictions_filtered['pred']
    )
#RMSE
eval_results['RMSE (>0)'] = np.sqrt(sklearn.metrics.mean_squared_error(
    df_predictions_filtered['target'], df_predictions_filtered['pred']
    ))
#R2
eval_results['R2 (>0)'] = sklearn.metrics.r2_score(
    df_predictions_filtered['target'], df_predictions_filtered['pred']
```

```
    )

eval_results = {key : round(eval_results[key], 2) for key in eval_results}
```

In [ ]:
```
# (>) denotes only fields with mean clover proportion above 0
pprint.pprint(eval_results)
```

```
{'MAE': 9.53,
 'MAE (>0)': 7.9,
 'R2': 0.48,
 'R2 (>0)': 0.62,
 'RMSE': 12.9,
 'RMSE (>0)': 10.27}
```

In [ ]:
```
plt.figure(figsize=(6,6))
sns.scatterplot(data=df_predictions[df_predictions.target >= 0.5], x='target', y='pr
                s=50, alpha=0.5, edgecolor=None)
plt.title(f'Cross validation results')
plt.xlabel('Measured Clover (%)')
plt.ylabel('Predicted Clover (%)')
plt.ylim(0, 60)
plt.xlim(0, 60)
plt.plot([0, 120], [0, 120], linestyle='dashed')
plt.show()
```



Feature Importance

In [ ]:
```
df_folds = pd.DataFrame(feature_sets)

# Calculate the mean importance for each feature across folds
mean_importances = df_folds.mean().sort_values(ascending=False)

# Sort the columns of the DataFrame according to the mean importance
df_folds_sorted = df_folds[mean_importances.index]

# Create the flipped violin plot with sorted features
plt.figure(figsize=(10, 8))
sns.violinplot(data=df_folds_sorted, orient='h', order=mean_importances.index, palet
plt.xlim(0, )
plt.xlabel('Gain')
plt.ylabel('Feature')
plt.title('Feature Importance ("gain") across farms (violin plot)', fontsize=21)
plt.show()
```