Changelog - August 1, 2023

Dane:

- Updated README
- Removed unused files
- Aggregation by harvest
- Feature importance
- Documentation and refactor
    - Dataset
    - Models (kinda-ish)
    - Copernicus
    - Soil moisture
    - Shared
    - Soil
    - Weather station

Daniel:

- Experimenting the ML models on the newly created datasets (downgrade).
- Create notebook pipeline for all models.
- Documentation

Dharmit:

- Worked on ML models(KNN, SVM)
- Experimented models on created dataset
- Ergot Documentation
- Model Documentation

Joseff:

- Documentation
- Model Pipeline Notebook
- Notebooks with Models grouped by class

Documentation

1. What data do we have?

2. How do we use it make a model?
   a. Notebook interface (WIP)

3. How do we update data?

4. Computer broke how do we rebuild the project?

What data do we have?

README

- Thanks Rob! (data dictionaries)
- Vertical and Horizontal drop downs

ergot_sample_feat_eng

- Schema: public
- Columns: 10

Similarly to the ergot_sample table, ergot_sample_feat_eng contains all samples, both infected and diesease free, submited to the Canadian Harvest program by farmers to be tested for ergot. Of the original data, samples without a specified province and or district were discarded. The data is enhanced with additional engineered features.

▶ Vertical view ergot_sample_feat_eng attribute list

▼ Horizontal view ergot_sample_feat_eng attribute list

	sample_id	year	province	crop_district	district	incidence	severity	dow
description	unique sample identifier		province abbreviation	non-unique identifier for a district within a province	unique region identifier	truth value for the presence of ergot	percentage of severity detected	com to e selli thre of 0
type	int	int	string	int	int	boolean	double	boo
unit								
constraints								

What data do we have?

Feature importance:

Data aggregated by month
[mean]|[minMax]|[straified on has_ergot]
2:max_rel_humid
5:min_dew_point_temp 7:max_temp
7:min_rel_humid 7:max_rel_humid
7:mean_rel_humid
8:max_dew_point_temp
8:mean_dew_point_temp
8:max_rel_humid 12:max_temp
avg_total_silt avg_percnt_carbon
avg_calcium_ph
avg_water_reten_10
avg_bulk_density avg_percnt_wood

Data aggregated by week
[mean]|[minMax]|[straified on has_ergot]
4:min_temp 4:min_dew_point_temp
4:mean_dew_point_temp 4:max_rel_humid
4:mean_rel_humid 7:max_temp
8:max_dew_point_temp 8:max_rel_humid
13:max_temp 13:min_rel_humid
20:min_dew_point_temp 21:min_temp
21:max_temp 21:mean_temp 24:min_rel_humid
24:max_stn_press 24:mean_stn_press
25:min_temp 25:min_dew_point_temp
25:mean_dew_point_temp 27:mean_rel_humid
28:max_temp 28:mean_rel_humid 29:max_temp
29:mean_temp 29:min_rel_humid
29:mean_rel_humid 30:max_temp 30:mean_temp
30:min_rel_humid 30:mean_rel_humid
31:max_temp 31:mean_temp
31:min_dew_point_temp 31:min_rel_humid
31:max_rel_humid 31:mean_rel_humid
33:min_temp 33:max_temp 33:min_rel_humid
34:min_temp 34:max_temp 34:mean_temp
34:min_dew_point_temp 34:max_dew_point_temp
34:mean_dew_point_temp 34:max_humidex
34:min_rel_humid 34:max_rel_humid
34:mean_rel_humid 35:min_dew_point_temp
36:min_dew_point_temp
36:mean_dew_point_temp 39:min_rel_humid
47:mean_rel_humid 52:max_temp
avg_total_sand avg_total_silt
avg_total_clay avg_percnt_carbon
avg_calcium_ph avg_proj_ph
avg_water_reten_0 avg_water_reten_10
avg_water_reten_33 avg_bulk_density
avg_percnt_wood avg_water_area

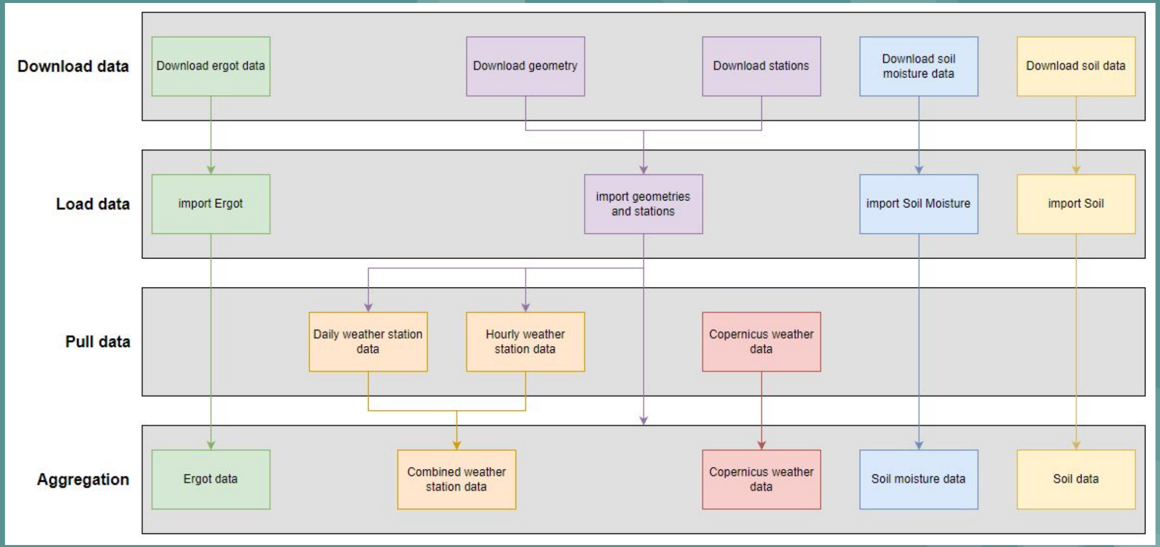
Quite a few similarities...

How do we update data?

loadData.ipynb

- Step by step:
  1. Add data folders
  2. Download data source files
  3. Pull data
  4. Aggregate data
  5. Create datasets

Steps to update the data in the future are documented as well



Prepares everything for models
Documentation on how to make changes to pull future data

Computer broke how do we rebuild the project

```
# ------------------------------------------------------------
# evaluator.py
#
# The purpose of the provided code is to define a class which contains methods for evaluating machine learning models for both classification
# and regression tasks.
#
# The class provides functions to calculate various evaluation metrics for the models, such as accuracy, R-squared, precision, recall, F1 score, and area under the
# receiver operating characteristic curve (AUC-ROC).
#
# Evaluation metrics:
# ------------------------------------------------------------
# Avg_accuracy: accuracy of stratified k fold cross validation using test data set (Perfect = 100)
# ------------------------------------------------------------
# R2: approximately how much of the observed variation can be explained by the model's inputs? (Perfect = 1)
# ------------------------------------------------------------
# Loss: summation of errors in our model (Perfect = 0)
# ------------------------------------------------------------
# Precision: the ability to classify positive samples in the model (Perfect = 1)
# ------------------------------------------------------------
# Recall: how many positive samples were correctly classified by the model (Perfect = 1)
# ------------------------------------------------------------
# F1: harmonic mean of precision and recall (Perfect = 1)
# ------------------------------------------------------------
# Auc: the ability to distinguish between all the Positive and the Negative class points (Perfect = 1)
# ------------------------------------------------------------
# neg_mean_squared_error: mean squared logarithmic summation of errors in our model (Perfect = 0)
# ------------------------------------------------------------
#
# Remarks:
# - Avg_accuracy is a bad measure when working with unbalanced datasets
# - Auc is really good when working with True and False classes
# - Further evaluation metric documentation can be found [here](https://scikit-learn.org/stable/modules/model_evaluation.html)
# ------------------------------------------------------------
```

Computer broke how do we rebuild the project

1. Purpose
2. Pseudocode
3. Tables
4. Functions used
5. Typical usage
6. Remarks

Purpose:
Reshapes columns of data by date (day, week or month)

Pseudocode:
- Calculate the year range
- Gather all the unique districts
- Collect the aggregated column names in a list
- Remove the irrelevant columns (these are the columns we wont want to appear once our data has been reshaped)
- Collect the row of data that is relevant to the current date, district combination
- Grab all attributes and establish them as key in a dictionary i.e {"DATE:attribute": value}
- Once finished for the current date, district combination, store the dictionary into a list

Remark: for this function to work correctly the following columns must be present given their dateType
- dateType=dates: year, district and month, day
- dateType=weeks: year, district and week
- dateType=months: year, district and month

Also note that we use a list of dictionaries since it is much faster to do so as opposed to the number of DataFrame manipulations we'd require otherwise

- Emphasis on linking relevant data
- Emphasis on requirements and output

Our pipeline notebooks

1. ETL (Extract, transform, load) the raw data
2. Select the tables (attributes) we want to experiment on - further aggregation could be done here to make sure we get what we want.
3. Then, split the dataset into testing and training dataset.
4. Balancing the data - avoid bias.
5. Normalize the data
6. Encode Categorical for any columns that needs to be converted to categorical using one-hot encoding technique
7. Pick the model (init, train, test, gather results)

Model Training

Dataset - Build a dataset for training. In this project the tables come from an SQL Database or CSV files

Most ML models that use supervised training methods have a list of attributes used for predicting and one column as a target variable that can be a class for classification models or a number for regression models

Year	District	5:MaxTemp	Ergot
1995	4610	30	0.04
1996	4610	26	0.05
1997	4610	24	0.03

Model Training

Splitting - Datasets can be split into training, validation, and test set. multiple ways depending on the validation method. In this project we used kfold and temporal splitting to name a few.

Splitter Classes

model_selection.GroupKFold([n_splits])	K-fold iterator variant with non-overlapping groups.
model_selection.GroupShuffleSplit([...])	Shuffle-Group(s)-Out cross-validation iterator
model_selection.KFold([n_splits, shuffle, ...])	K-Folds cross-validator
model_selection.LeaveOneGroupOut()	Leave One Group Out cross-validator
model_selection.LeavePGroupsOut(n_groups)	Leave P Group(s) Out cross-validator
model_selection.LeaveOneOut()	Leave-One-Out cross-validator
model_selection.LeavePOut(p)	Leave-P-Out cross-validator
model_selection.PredefinedSplit(test_fold)	Predefined split cross-validator
model_selection.RepeatedKFold(*[, n_splits, ...])	Repeated K-Fold cross validator.
model_selection.RepeatedStratifiedKFold(*[, ...])	Repeated Stratified K-Fold cross validator.
model_selection.ShuffleSplit([n_splits, ...])	Random permutation cross-validator
model_selection.StratifiedKFold([n_splits, ...])	Stratified K-Folds cross-validator.
model_selection.StratifiedShuffleSplit([...])	Stratified ShuffleSplit cross-validator
model_selection.StratifiedGroupKFold([...])	Stratified K-Folds iterator variant with non-overlapping groups.
model_selection.TimeSeriesSplit([n_splits, ...])	Time Series cross-validator

Year	District	5:MaxTemp	Ergot
1995	4610	30	0.04
1996	4610	26	0.05
1997	4610	24	0.03

Model Training

Normalization / Scaling - Some models will require that the attributes are scaled, for example neural networks are biased by the magnitude of attribute values.

Year	District	5:MaxTemp	Ergot
1995	4610	1	-1
1996	4610	-1	1
1997	4610	-1	-1

sklearn.preprocessing: Preprocessing and Normalization

The sklearn.preprocessing module includes scaling, centering, normalization, binarization methods.

User guide: See the Preprocessing data section for further details.

preprocessing.Binarizer(*[, threshold, copy])	Binarize data (set feature values to 0 or 1) according to a threshold.
preprocessing.FunctionTransformer([func, ...])	Constructs a transformer from an arbitrary callable.
preprocessing.KBinsDiscretizer([n_bins, ...])	Bin continuous data into intervals.
preprocessing.KernelCenterer()	Center an arbitrary kernel matrix K.
preprocessing.LabelBinarizer(*[, neg_label, ...])	Binarize labels in a one-vs-all fashion.
preprocessing.LabelEncoder()	Encode target labels with value between 0 and n_classes-1.
preprocessing.MultiLabelBinarizer(*[, ...])	Transform between iterable of iterables and a multilabel format.
preprocessing.MaxAbsScaler(*[, copy])	Scale each feature by its maximum absolute value.
preprocessing.MinMaxScaler([feature_range, ...])	Transform features by scaling each feature to a given range.
preprocessing.Normalizer([norm, copy])	Normalize samples individually to unit norm.
preprocessing.OneHotEncoder(*[, categories, ...])	Encode categorical features as a one-hot numeric array.
preprocessing.OrdinalEncoder(*[, ...])	Encode categorical features as an integer array.
preprocessing.PolynomialFeatures([degree, ...])	Generate polynomial and interaction features.
preprocessing.PowerTransformer([method, ...])	Apply a power transform featurewise to make data more Gaussian-like.
preprocessing.QuantileTransformer(*[, ...])	Transform features using quantiles information.
preprocessing.RobustScaler(*[, ...])	Scale features using statistics that are robust to outliers.
preprocessing.SplineTransformer([n_knots, ...])	Generate univariate B-spline bases for features.
preprocessing.StandardScaler(*[, copy, ...])	Standardize features by removing the mean and scaling to unit variance.
preprocessing.TargetEncoder([categories, ...])	Target Encoder for regression and classification targets.

preprocessing.add_dummy_feature(X[, value])	Augment dataset with an additional dummy feature.
preprocessing.binarize(X, *[, threshold, copy])	Boolean thresholding of array-like or scipy.sparse matrix.
preprocessing.label_binarize(y, *, classes)	Binarize labels in a one-vs-all fashion.
preprocessing.maxabs_scale(X, *[, axis, copy])	Scale each feature to the [-1, 1] range without breaking the sparsity.
preprocessing.minmax_scale(X[, ...])	Transform features by scaling each feature to a given range.
preprocessing.normalize(X[, norm, axis, ...])	Scale input vectors individually to unit norm (vector length).
preprocessing.quantile_transform(X, *[, ...])	Transform features using quantiles information.
preprocessing.robust_scale(X, *[, axis, ...])	Standardize a dataset along any axis.
preprocessing.scale(X, *[, axis, with_mean, ...])	Standardize a dataset along any axis.
preprocessing.power_transform(X[, method, ...])	Parametric, monotonic transformation to make data more Gaussian-like.

Model Training

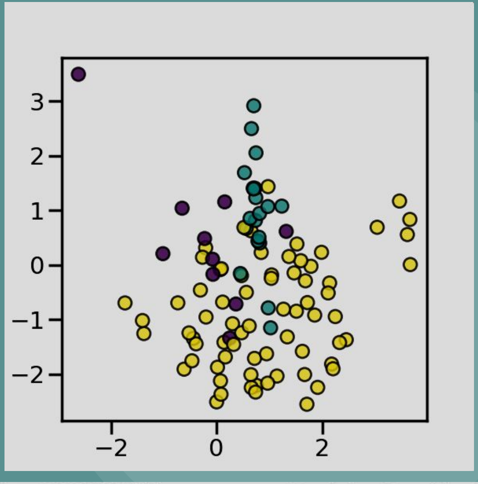
Balancing - Datasets can have very skewed classes, for example this project has < 3% of samples with ergot above the downgrade severity threshold. There are multiple methods of upsampling, downsampling, and combined methods. In this project, we used upsampling and SMOTENN.

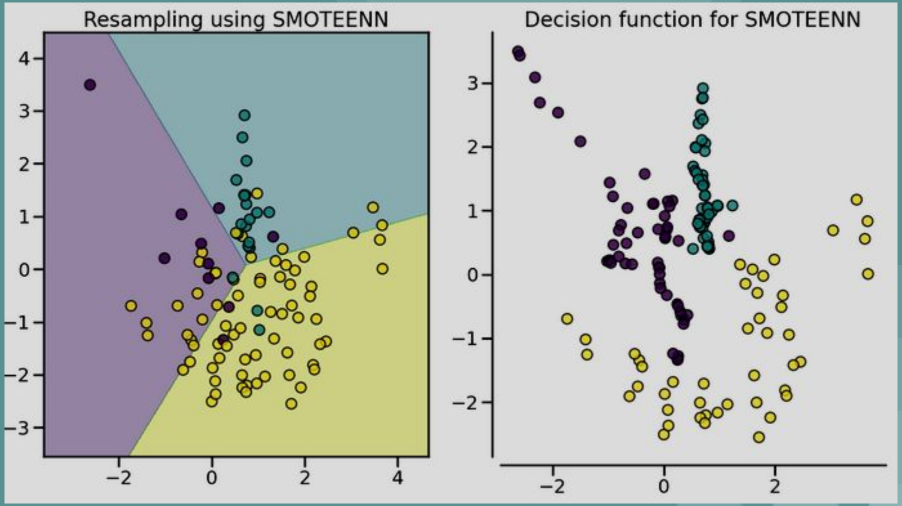
```
downgrade
False     122202
True        2082
Name: count, dtype: int64
downgrade
False      26307
True        1016
Name: count, dtype: int64
```

Train

Test

```
downgrade
False     115239
True       25156
Name: count, dtype: int64
```





Model Training

In this project we tried SVM, Random forest, decision trees, boost algo classifiers, and MLP

sklearn.ensemble: Ensemble Methods

The sklearn.ensemble module includes ensemble-based methods for classification, regression and anomaly detection.

User guide: See the Ensemble methods section for further details.

ensemble.AdaBoostClassifier([estimator, ...])	An AdaBoost classifier.
ensemble.AdaBoostRegressor([estimator, ...])	An AdaBoost regressor.
ensemble.BaggingClassifier([estimator, ...])	A Bagging classifier.
ensemble.BaggingRegressor([estimator, ...])	A Bagging regressor.
ensemble.ExtraTreesClassifier([...])	An extra-trees classifier.
ensemble.ExtraTreesRegressor([n_estimators, ...])	An extra-trees regressor.
ensemble.GradientBoostingClassifier(*[, ...])	Gradient Boosting for classification.
ensemble.GradientBoostingRegressor(*[, ...])	Gradient Boosting for regression.
ensemble.IsolationForest(*[, n_estimators, ...])	Isolation Forest Algorithm.
ensemble.RandomForestClassifier([...])	A random forest classifier.
ensemble.RandomForestRegressor([...])	A random forest regressor.
ensemble.RandomTreesEmbedding([...])	An ensemble of totally random trees.
ensemble.StackingClassifier(estimators[, ...])	Stack of estimators with a final classifier.
ensemble.StackingRegressor(estimators[, ...])	Stack of estimators with a final regressor.
ensemble.VotingClassifier(estimators, *[, ...])	Soft Voting/Majority Rule classifier for unfitted estimators.
ensemble.VotingRegressor(estimators, *[, ...])	Prediction voting regressor for unfitted estimators.
ensemble.HistGradientBoostingRegressor([...])	Histogram-based Gradient Boosting Regression Tree.
ensemble.HistGradientBoostingClassifier([...])	Histogram-based Gradient Boosting Classification Tree.

Ensemble methods

The imblearn.ensemble module include methods generating under-sampled subsets combined inside an ensemble.

Boosting algorithms

EasyEnsembleClassifier([n_estimators, ...])	Bag of balanced boosted learners also known as EasyEnsemble.
RUSBoostClassifier([estimator, ...])	Random under-sampling integrated in the learning of AdaBoost.

Bagging algorithms

BalancedBaggingClassifier([estimator, ...])	A Bagging classifier with additional balancing.
BalancedRandomForestClassifier([...])	A balanced random forest classifier.

XGBoost is an optimized distributed gradient boosting library designed to be highly **efficient**, **flexible** and **portable**. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. The same code runs on major distributed environment (Hadoop, SGE, MPI) and can solve problems beyond billions of examples.

Model Training

Feature Selection Tuning Optimization - There are multiple parts of the pipeline where optimizers and feature selectors can be used to improve models.

sklearn.feature_selection: Feature Selection

The sklearn.feature_selection module implements feature selection algorithms. It currently includes univariate filter selection met
rithm.

User guide: See the Feature selection section for further details.

feature_selection.GenericUnivariateSelect([...])	Univariate feature selector with configurable strategy.
feature_selection.SelectPercentile([...])	Select features according to a percentile of the highest scores.
feature_selection.SelectKBest([score_func, k])	Select features according to the k highest scores.
feature_selection.SelectFpr([score_func, alpha])	Filter: Select the pvalues below alpha based on a FPR test.
feature_selection.SelectFdr([score_func, alpha])	Filter: Select the p-values for an estimated false discovery rate.
feature_selection.SelectFromModel(estimator, *)	Meta-transformer for selecting features based on importance weights.
feature_selection.SelectFwe([score_func, alpha])	Filter: Select the p-values corresponding to Family-wise error rate.
feature_selection.SequentialFeatureSelector(...)	Transformer that performs Sequential Feature Selection.
feature_selection.RFE(estimator, *[, ...])	Feature ranking with recursive feature elimination.
feature_selection.RFECV(estimator, *[, ...])	Recursive feature elimination with cross-validation to select features.
feature_selection.VarianceThreshold([threshold])	Feature selector that removes all low-variance features.

feature_selection.chi2(X, y)	Compute chi-squared stats between each non-negative feature and class.
feature_selection.f_classif(X, y)	Compute the ANOVA F-value for the provided sample.
feature_selection.f_regression(X, y, *[, ...])	Univariate linear regression tests returning F-statistic and p-values.
feature_selection.r_regression(X, y, *[, ...])	Compute Pearson's r for each features and the target.
feature_selection.mutual_info_classif(X, y, *)	Estimate mutual information for a discrete target variable.
feature_selection.mutual_info_regression(X, y, *)	Estimate mutual information for a continuous target variable.

Hyper-parameter optimizers

model_selection.GridSearchCV(estimator, ...)	Exhaustive search over specified parameter values for an estimator.
model_selection.HalvingGridSearchCV(...[, ...])	Search over specified parameter values with successive halving.
model_selection.ParameterGrid(param_grid)	Grid of parameters with a discrete number of values for each.
model_selection.ParameterSampler(...[, ...])	Generator on parameters sampled from given distributions.
model_selection.RandomizedSearchCV(...[, ...])	Randomized search on hyper parameters.
model_selection.HalvingRandomSearchCV(...[, ...])	Randomized search on hyper parameters.

FINAL PRESENTATIONS

Friday, August 18th, 2023