2020 FORCE ML COMPETITION

How to predict lithology based on well logs using ML

The problem

Company

FORCE is a cooperating forum for improved exploration and improved oil and gas recovery conducted by oil and gas companies and authorities in Norway.

Context

All E&P companies must interpret lithologies from well logs to know where to find hydrocarbons.

This process is laborious and not scalable. 2-3 per well. Basin studies require hundreds of wells.

Problem statement

How can E&P companies improve the lithology classification score to less than -0.52 in less than five weeks?

Challenges deep-dive

Challenge 1

Incomplete well logs

Acquiring some logs is very expensive.

There are many missing values in the logs.

Challenge 2

Large data set

There are nearly 1.2 M samples in the train set.

Have to be smart handling RAM.

Challenge 3

Cost-sensitive classifier

The cost of misclassifications is not the same for all lithologies.

Solution

Improve the winning submission

I refactored the winning submission code to understand its intricacies and assumptions.

Then, I explore alternatives to some of the modeling decisions made.

Implementation

The data

Train:

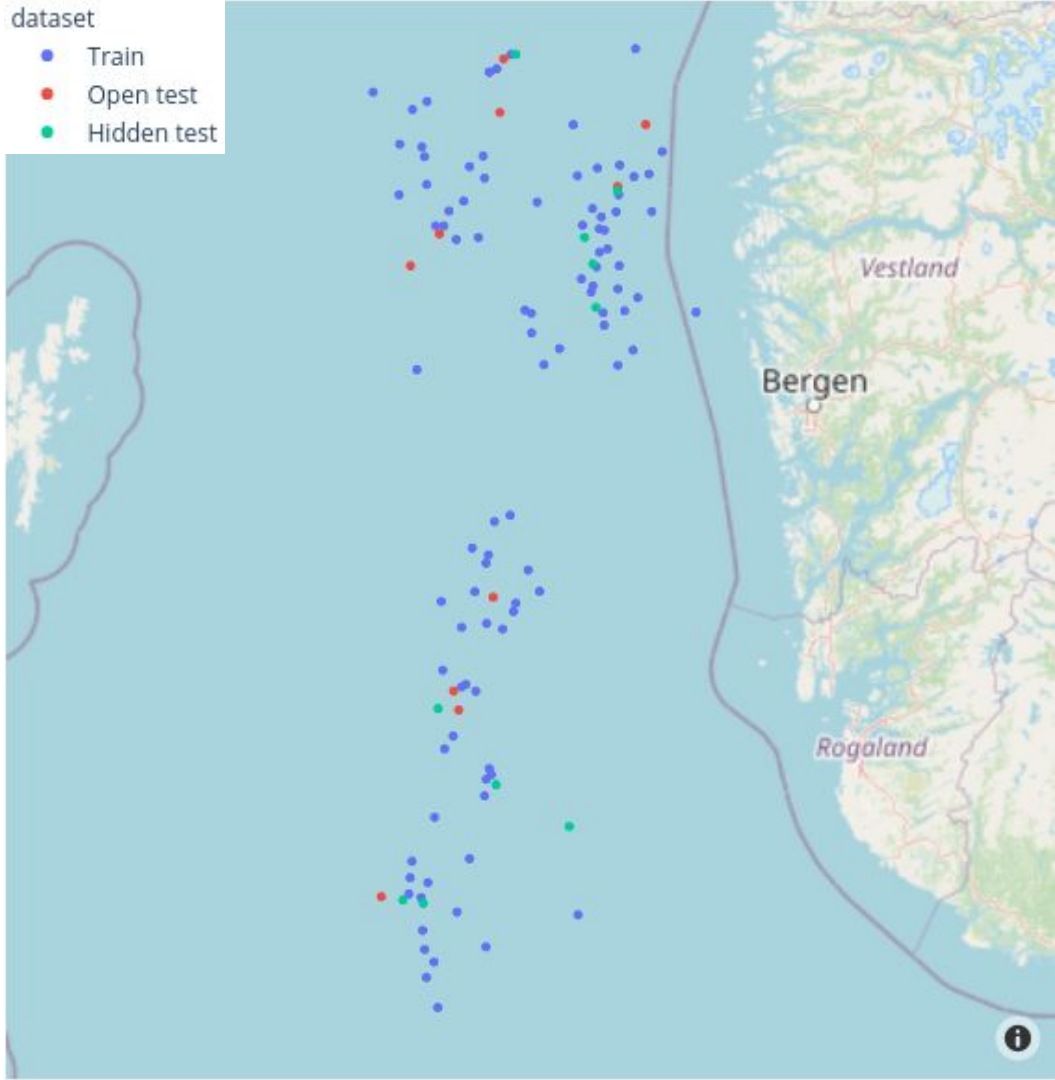
- 98 well.
- 12 lithology classes.
- 28 features, 8 with more than 50% missing.

Open test:

- 10 wells.
- Feature distribution similar to Train.

Hidden test:

- 10 wells.
- Feature distribution similar to Train.



The Scoring Function

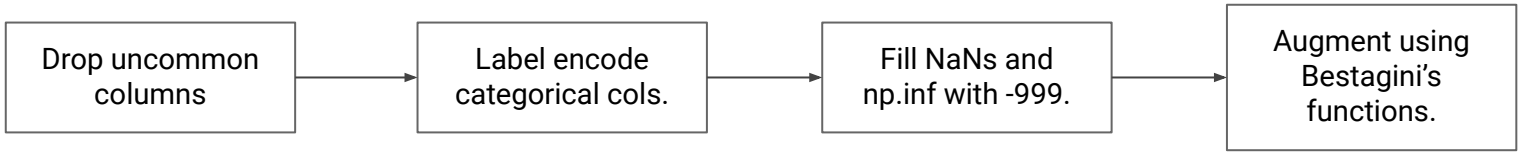
The scoring function was designed to weight misclassification errors differently

$$S = -\frac{1}{N}\sum_{i=0}^{N} A_{\hat{y}_i y_i}$$

N is the number of samples
A is the penalty matrix
$\hat{y}_i$ is the true target for sample *i*
y_i is the prediction for sample *i*

label \ prediction	Sandstone	Sandstone/Shale	Shale	Marl	Dolomite	Limestone	Chalk	Halite	Anhydrite	Tuff	Coal	Crystalline Basement
Sandstone	0	2	3.5	3	3.75	3.5	3.5	4	4	2.5	3.875	3.25
Sandstone/Shale	2	0	2.375	2.75	4	3.75	3.75	3.875	4	3	3.75	3
Shale	3.5	2.375	0	2	3.5	3.5	3.75	4	4	2.75	3.25	3
Marl	3	2.75	2	0	2.5	2	2.25	4	4	3.375	3.75	3.25
Dolomite	3.75	4	3.5	2.5	0	2.625	2.875	3.75	3.25	3	4	3.625
Limestone	3.5	3.75	3.5	2	2.625	0	1.375	4	3.75	3.5	4	3.625
Chalk	3.5	3.75	3.75	2.25	2.875	1.375	0	4	3.75	3.125	4	3.75
Halite	4	3.875	4	4	3.75	4	4	0	2.75	3.75	3.75	4
Anhydrite	4	4	4	4	3.25	3.75	3.75	2.75	0	4	4	3.875
Tuff	2.5	3	2.75	3.375	3	3.5	3.125	3.75	4	0	2.5	3.25
Coal	3.875	3.75	3.25	3.75	4	4	4	3.75	4	2.5	0	4
Crystalline Basement	3.25	3	3	3.25	3.625	3.625	3.75	4	3.875	3.25	4	0

The winning submission preprocess



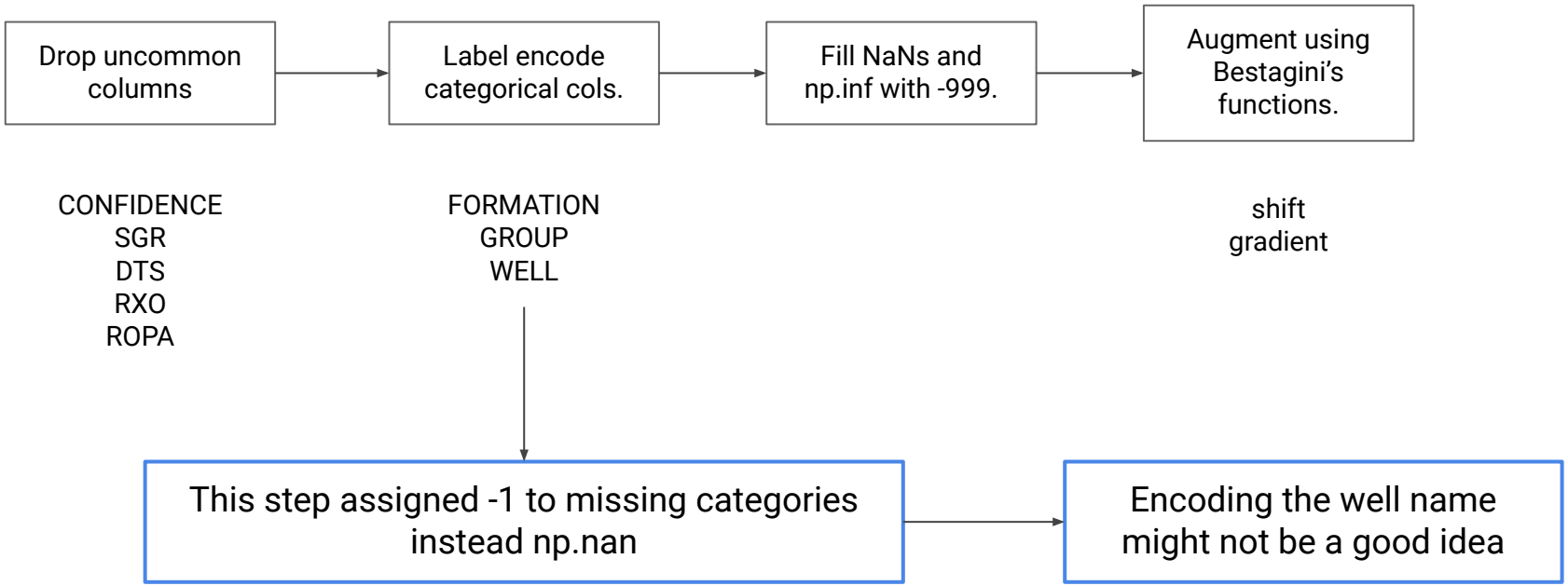
CONFIDENCE
SGR
DTS
RXO
ROPA

FORMATION
GROUP
WELL

shift
gradient

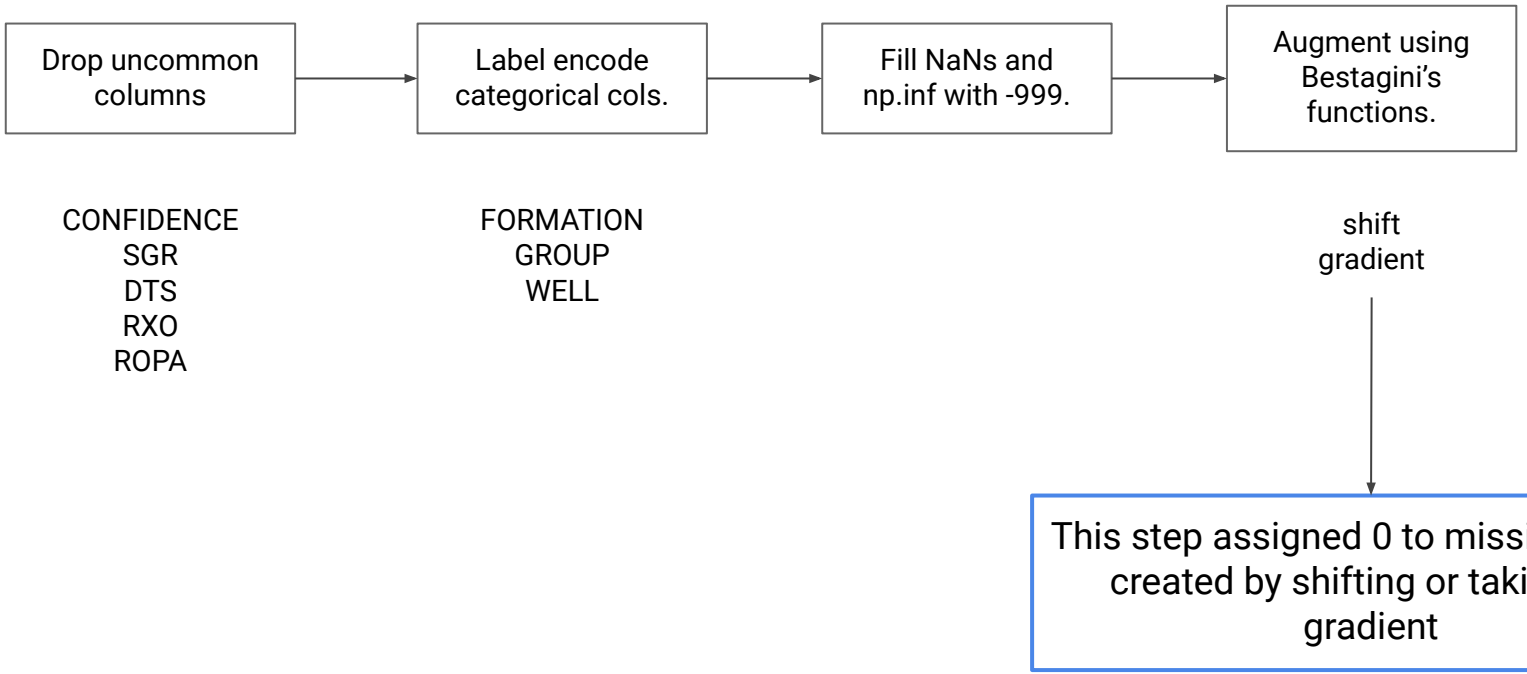
The winning submission preprocess

Revised



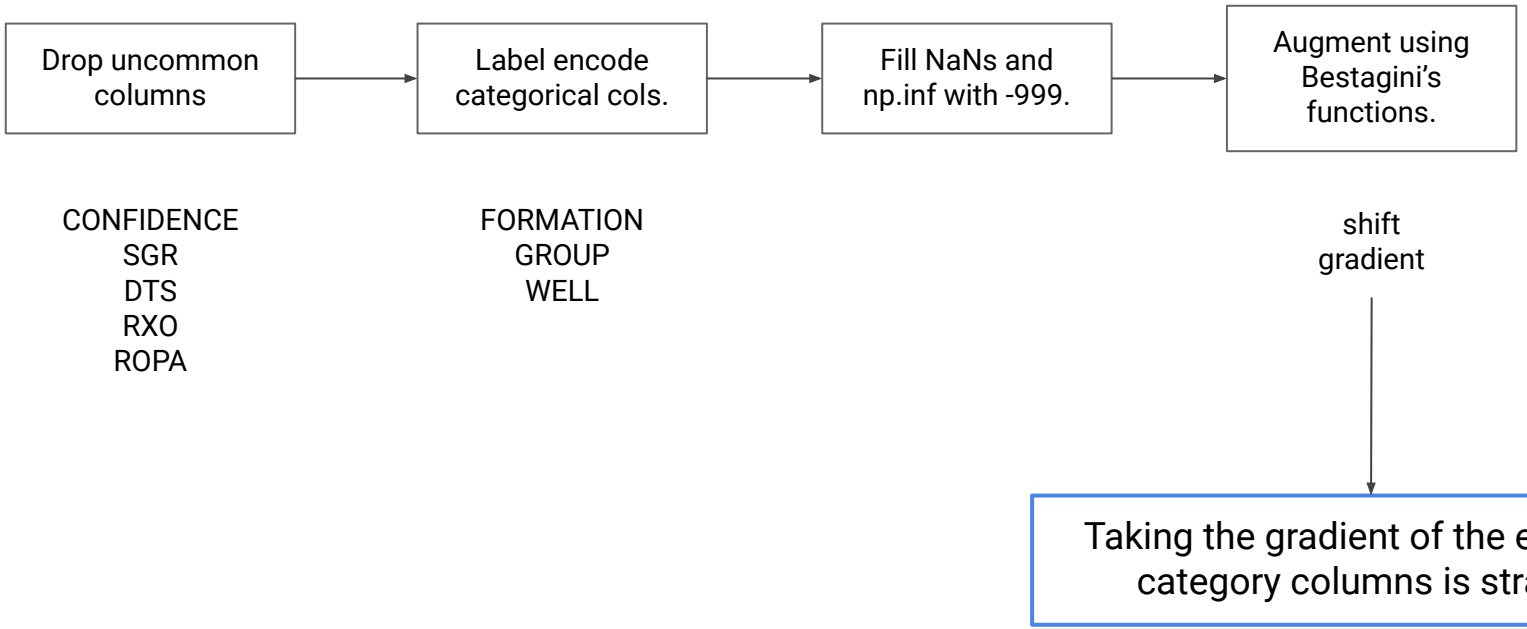
The winning submission preprocess

Revised

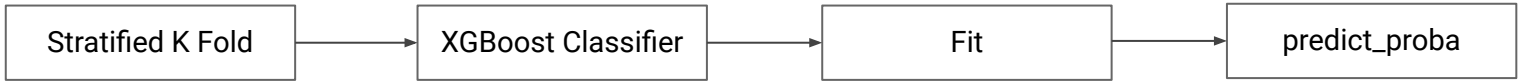


The winning submission preprocess

Revised



The winning submission model



10 splits
Shuffle is True

n_stimator 100
max_depth 10
multi:softprob
gpu_hist

early_stopping_rounds 100
eval_set (X_test, y_test)

X_test

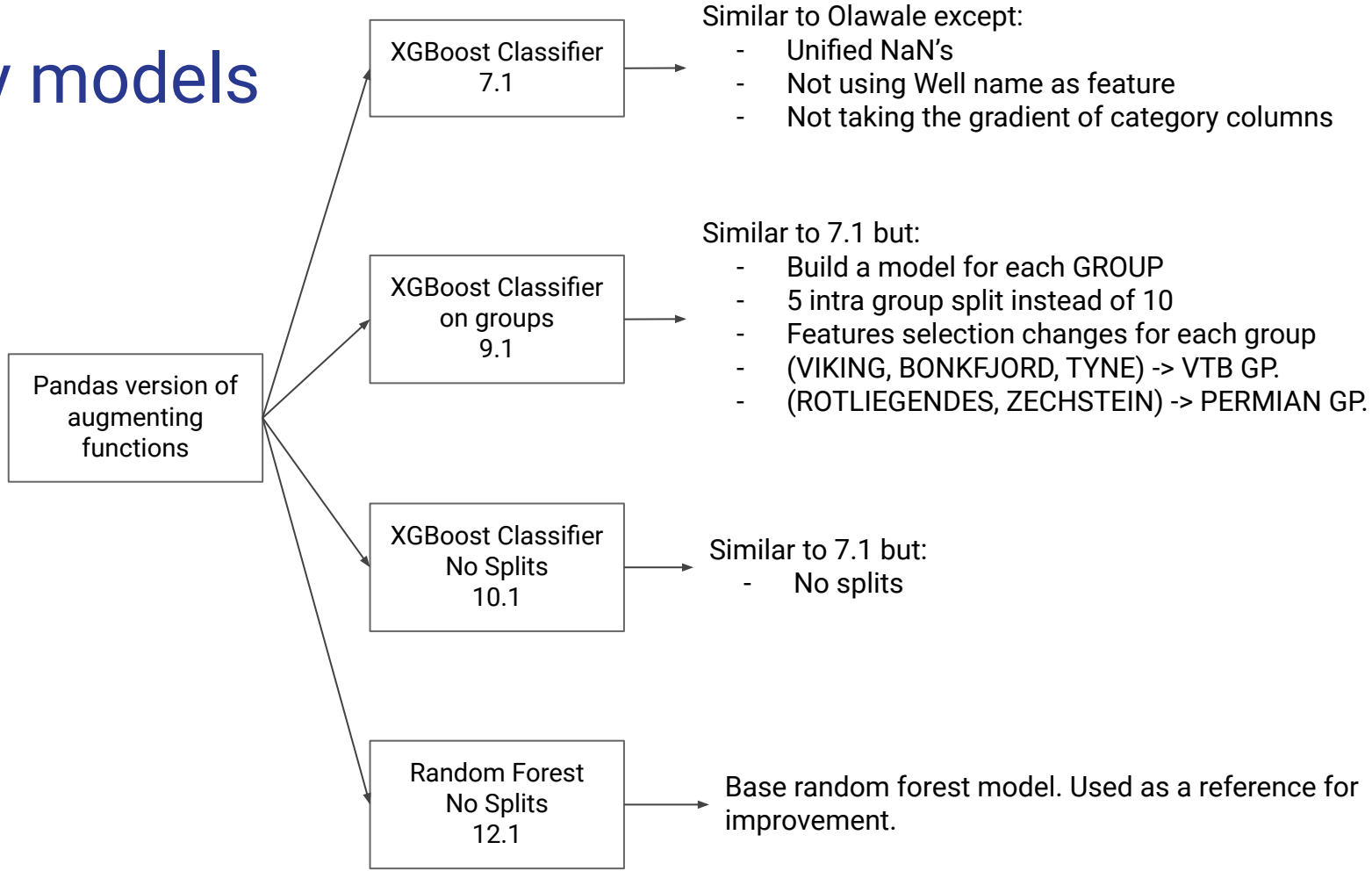
There will be 10 models, one for each data split.

The predicted probabilities are averaged across models.

The largest average probability defines the predicted class

Open score: -0.515
Hidden score: -0.471

My models


Pandas version of augmenting functions
XGBoost Classifier 7.1
Similar to Olawale except:
- Unified NaN's
- Not using Well name as feature
- Not taking the gradient of category columns
XGBoost Classifier on groups 9.1
Similar to 7.1 but:
- Build a model for each GROUP
- 5 intra group split instead of 10
- Features selection changes for each group
- (VIKING, BONKFJORD, TYNE) -> VTB GP.
- (ROTLIEGENDES, ZECHSTEIN) -> PERMIAN GP.
XGBoost Classifier No Splits 10.1
Similar to 7.1 but:
- No splits
Random Forest No Splits 12.1
Base random forest model. Used as a reference for improvement.

My models scores

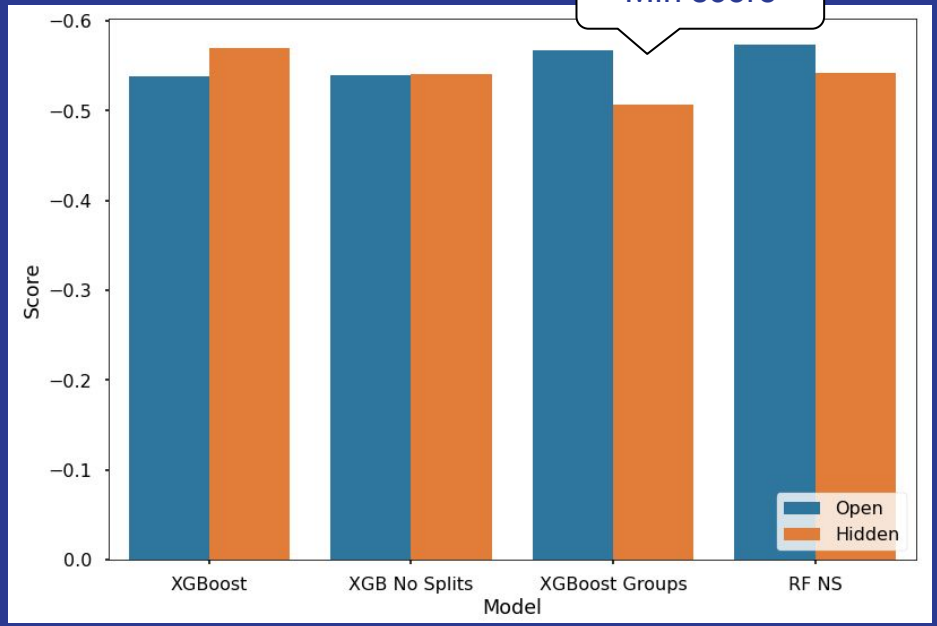
XGBoost Groups outperforms in the hidden test

Model	Notebook	Open Score	Hidden Score
XGBoost	7.0, 7.1, 7.2	-0.538	-0.570
XGBoost No Splits	10.0, 10.1, 10.2	-0.539	-0.541
XGBoost Groups	9.0, 9.1, 9.2	-0.567	-0.506
Random Forest NS	12.0, 12.1, 12.2	-0.574	-0.542

My models

-0.506 Hidden Score
XGBoost Grouped

Would have ranked 5th

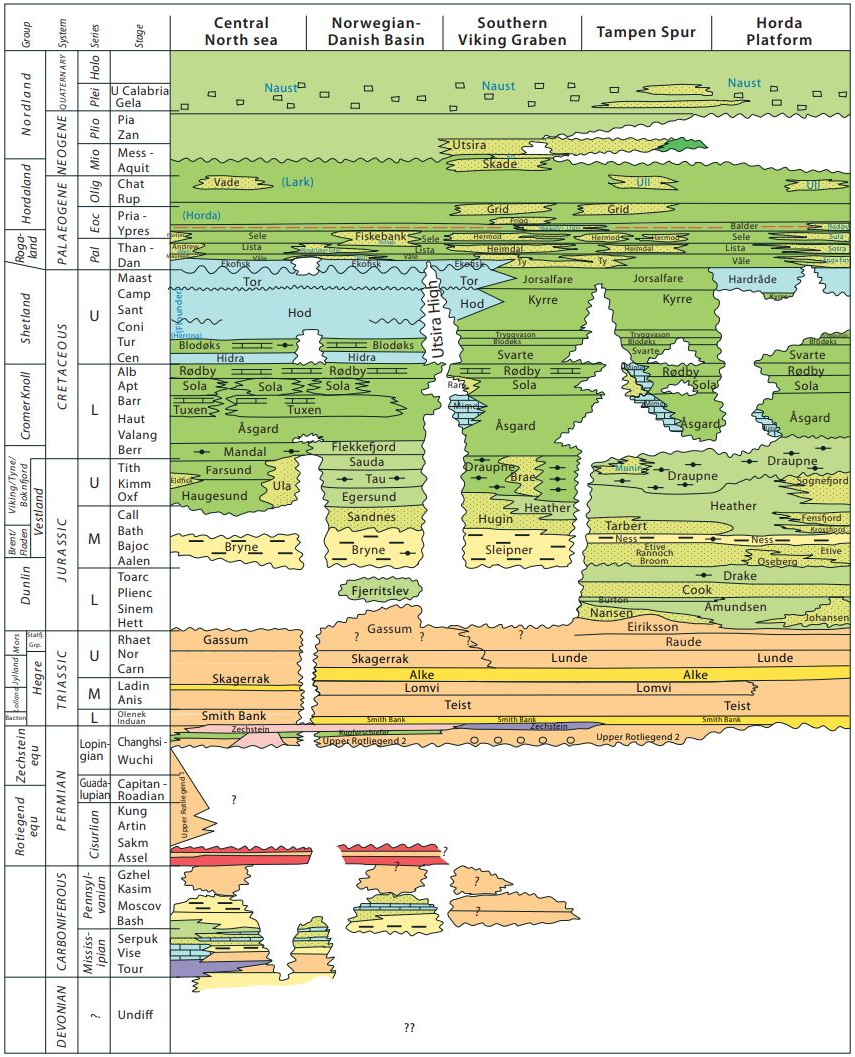


Final Score	My team name /personal name is	Score on hidden dataset	My current score XEEK leader board is	My current position XEEK leader board is
1	**Olawale Ibrahim**	**-0.469**	-0.5118	24
2	**GIR TEAM**	**-0.4792**	-0.5037	11
3	**Lab.ICA-Team / Smith A.**	**-0.49536**	-0.4943	6
4	H3G (Haoyuan Zhang, Harry Brandsen, Gregory Barrere, Helena Nandi Formentin)	**-0.504489**	-0.509	17
5	ISPL Team	**-0.50835**	-0.4885	2
6	Jiampiers C.	**-0.50886**	-0.5014	9
7	José Bermúdez	**-0.509061**	-0.5052	14
8	Bohdan Pavlyshenko	**-0.51713**	-0.5112	22
9	Jeremy Zhao	**-0.51733**	0.5264	31
10	Campbell Hutcheson	**-0.52206**	-0.505	13

The Groups

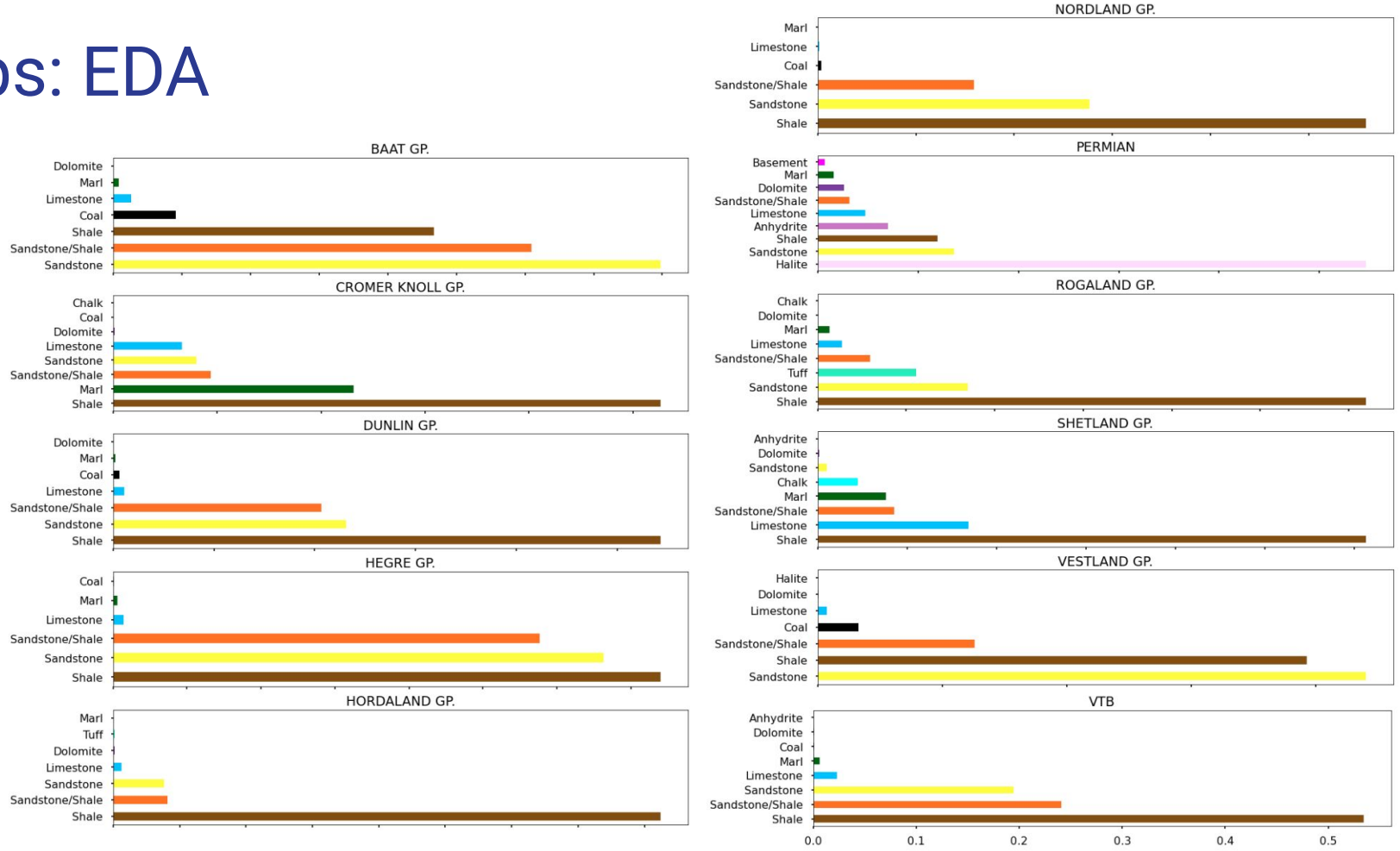
- Norwegian North Sea lithologies.
- Not all lithologies are present in all groups
- This inspired me to split the data by groups.
- Also the groups feature has no missing values in the train or test sets.

https://www.npd.no/globalassets/1-npd/fakta/geologi-eng/ns-od1409001.pdf



Groups: EDA

Lith presence per group



Groups: EDA

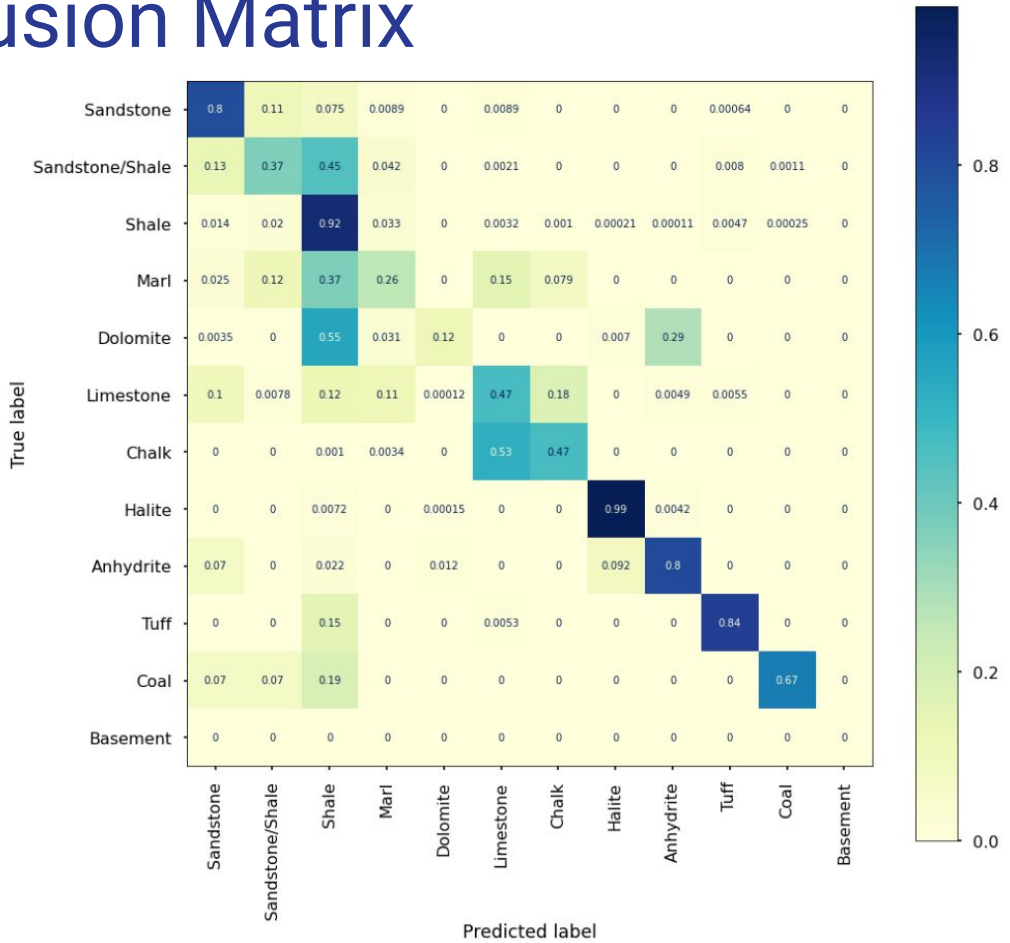
Log presence per group

- Percent of samples per well
- Poor availability:
  - SGR
  - DTS
  - DCAL
  - RMIC
  - ROPA
  - RXO
- Only in a few groups:
  - ROP
  - MUDWEIGHT

	BAAT GP.	CROMER KNOLL GP.	DUNLIN GP.	HEGRE GP.	HORDALAND GP.	NORDLAND GP.	PERMIAN	ROGALAND GP.	SHETLAND GP.	VESTLAND GP.	VTB
WELL	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
DEPTH_MD	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
X_LOC	1.000000	0.985168	1.000000	0.998778	0.998765	0.984133	0.694278	1.000000	0.995125	0.976643	0.997997
Y_LOC	1.000000	0.985168	1.000000	0.998778	0.998765	0.984133	0.694278	1.000000	0.995125	0.976643	0.997997
Z_LOC	1.000000	0.985168	1.000000	0.998778	0.998765	0.984133	0.694278	1.000000	0.995125	0.976643	0.997997
GROUP	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
FORMATION	1.000000	1.000000	1.000000	1.000000	0.813641	0.272724	1.000000	1.000000	1.000000	1.000000	1.000000
CALI	0.995924	0.870260	0.970240	0.999712	0.908693	0.827240	0.999468	0.880768	0.949527	1.000000	0.981053
RSHA	0.748569	0.564163	0.634757	0.532308	0.501448	0.491712	0.364072	0.544390	0.432717	0.739776	0.671647
RMED	0.954024	0.975956	0.973767	0.998778	0.984152	0.933241	0.347305	0.985130	0.985886	0.974575	0.971275
RDEP	0.998772	0.985168	0.999496	0.998778	0.998765	0.984133	0.694278	1.000000	0.995125	0.976643	0.997000
RHOB	0.998102	0.963169	0.989881	0.999928	0.802487	0.523805	0.735196	0.917094	0.885245	0.999387	0.970380
GR	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
SGR	0.195070	0.056881	0.134081	0.072235	0.022285	0.043529	0.000000	0.026117	0.025988	0.099403	0.138687
NPHI	0.998995	0.877733	0.989680	0.990656	0.434787	0.208673	0.741184	0.553424	0.682995	0.998660	0.962767
PEF	0.620216	0.593005	0.609649	0.555308	0.543310	0.423643	0.106387	0.597003	0.648833	0.757314	0.583816
DTC	0.984535	0.978192	0.940681	0.952275	0.922870	0.824827	0.982635	0.950646	0.929376	0.969176	0.971283
SP	0.707730	0.534002	0.749549	0.821965	0.830567	0.852668	0.303526	0.829958	0.599163	0.706655	0.724996
BS	0.521844	0.607798	0.620364	0.800331	0.528993	0.491452	0.682635	0.575138	0.599253	0.757505	0.668324
ROP	0.100969	0.592087	0.276601	0.454539	0.523010	0.614369	0.999601	0.416646	0.449980	0.752029	0.326035
DTS	0.186026	0.291036	0.134156	0.128944	0.041510	0.017625	0.063273	0.104938	0.265062	0.186705	0.286902
DCAL	0.041147	0.189469	0.267893	0.293538	0.328430	0.241501	0.210645	0.331095	0.196592	0.562031	0.151956
DRHO	0.966753	0.966858	0.971659	0.999928	0.758554	0.531106	0.736327	0.887983	0.876861	0.999464	0.967182
MUDWEIGHT	0.000000	0.252924	0.173162	0.293467	0.409527	0.398583	0.915902	0.303576	0.133467	0.529714	0.108788
RMIC	0.177902	0.313016	0.159088	0.095378	0.098562	0.000000	0.000000	0.188845	0.140398	0.222163	0.298308
ROPA	0.155905	0.380753	0.106193	0.137210	0.073367	0.059593	0.021490	0.086726	0.336417	0.123219	0.222968
RXO	0.353683	0.207798	0.325524	0.249048	0.250407	0.229895	0.105788	0.317589	0.236420	0.438773	0.383543
20_LITHOFACIES_LITHOLOGY	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
0_LITHOFACIES_CONFIDENCE	0.999972	0.999560	0.999924	0.998850	0.999881	0.999776	0.999933	0.999932	0.999885	0.999847	0.999831
GROUPED	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

XGBoost Groups: Confusion Matrix

- Sandstone/Shale is a mixed bag
- It does very well at predicting:
  - Shale
  - Halite
  - Anhydrite
- Getting confused with Shale:
  - Marl
  - Dolomite
  - Tuff
  - Coal
- Hard to separate Chalk from Limestone
- No Basement on Hidden Test



Future work

- Build category logs comparison
- Explore differences between my version of Olawale XBoost and the original submission
  - Including the cat. column gradient
  - Including the well name column
- Tune model hyperparameters
- Use cost sensitive approach
  - MetaCost algorithm needs memory efficient implementation
  - Add custom score function to XGBoost train step

Memory trap

```
def fit(self, flag, num_class):
    """
    :param flag: The name of classification labels
    :param num_class: The number of classes
    :return: Classifier
    """
    col = [col for col in self.S.columns if col != flag]
    S_ = {}
    M = []

    for i in range(self.m):
        # Let S_[i] be a resample of S with self.n examples
        S_[i] = self.S.sample(n=self.n, replace=True)

        X = S_[i][col].values
        y = S_[i][flag].values

        # Let M[i] = model produced by applying L to S_[i]
        model = clone(self.L)
        M.append(model.fit(X, y))

    label = []
    S_array = self.S[col].values
    for i in range(len(self.S)):
        if not self.q:
            k_th = [k for k, v in S_.items() if i not in v.index]
            M_ = list(np.array(M)[k_th])
        else:
            M_ = M

        if self.p:
            P_j = [model.predict_proba(S_array[[i]]) for model in M_]
        else:
            P_j = []
            vector = [0] * num_class
            for model in M_:
                vector[model.predict(S_array[[i]])] = 1
                P_j.append(vector)

        # Calculate P(j|x)
        P = np.array(np.mean(P_j, 0)).T

        # Relabel
        label.append(np.argmin(self.C.dot(P)))

    # Model produced by applying L to S with relabeled y
    X_train = self.S[col].values
    y_train = np.array(label)
    model_new = clone(self.L)
    model_new.fit(X_train, y_train)

    return model_new
```