Glitch Classification Data Challenge

Roberto Corizzo
University of Bari Aldo Moro

Data Science School, Braga 25-27.03.2019

G2NET

COST EUROPEAN COOPERATION IN SCIENCE AND TECHNOLOGY

Overview

- Ensemble models
- Auto-encoders
- Notebooks on Random Forest, XGBoost, Keras

Data Challenge: Tomorrow

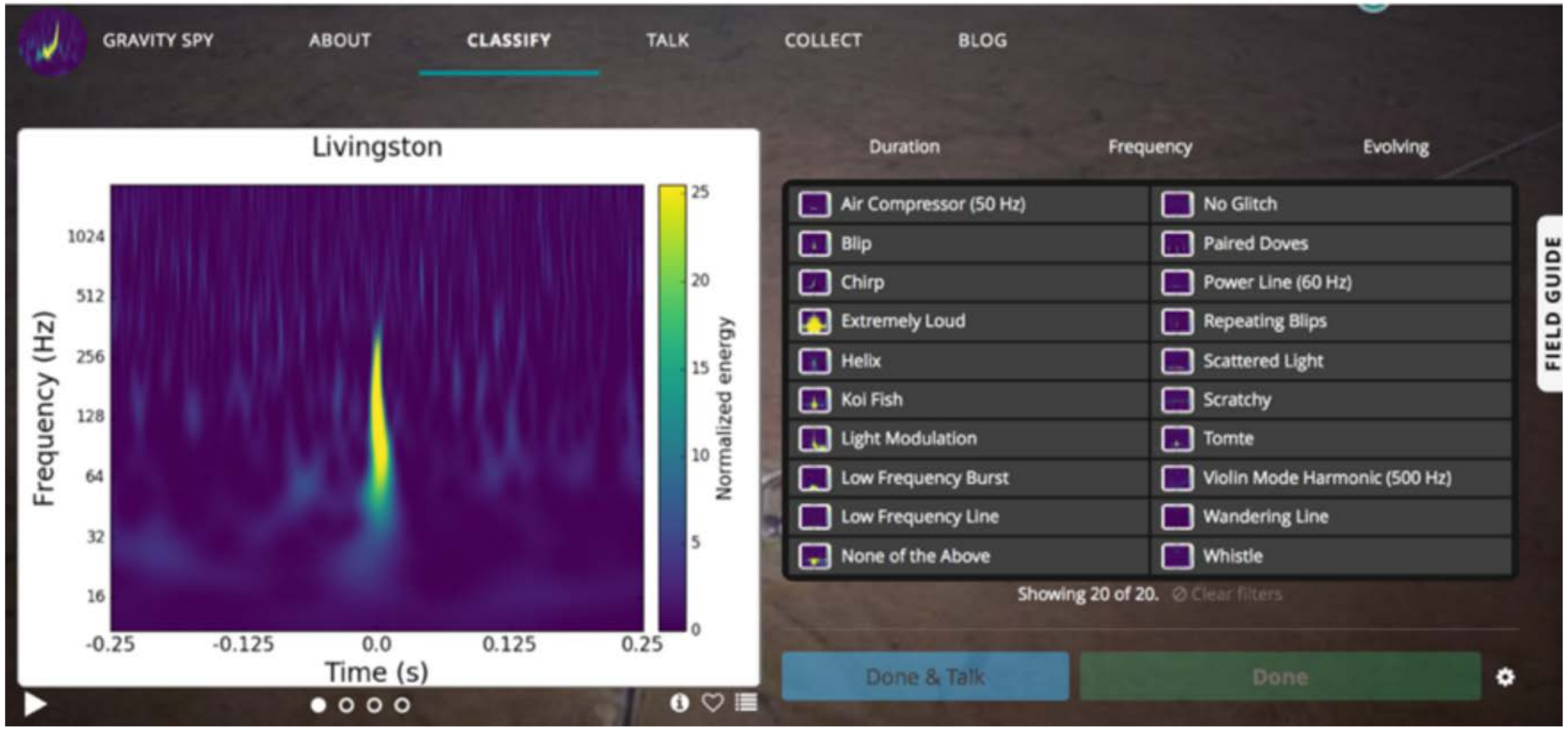
Filip’s presentation on time series data
Tutorial on 1D CNN and 2D CNN notebooks

Note about the time series glitch dataset:

- Some time series present missing values (NaN)
- Those time series were not supposed to be included in the dataset and can be discarded

Glitch classification: Part I

gravityspy.org Zevin et al, 2017, CQG



Air Compressor (50 Hz)	No Glitch
Blip	Paired Doves
Chirp	Power Line (60 Hz)
Extremely Loud	Repeating Blips
Helix	Scattered Light
Koi Fish	Scratchy
Light Modulation	Tomte
Low Frequency Burst	Violin Mode Harmonic (500 Hz)
Low Frequency Line	Wandering Line
None of the Above	Whistle

Showing 20 of 20. Clear filters

Dataset specifications

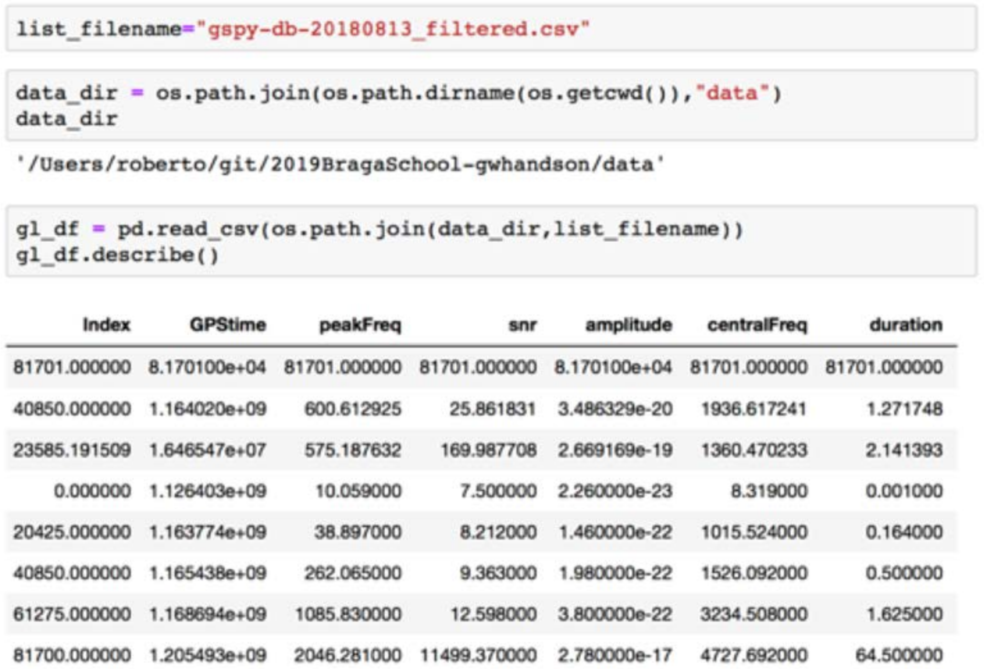
6667 glitches seen by LIGO detectors during O1

- **Numeric features**
  - GPStime
  - peakFreq
  - snr
  - centralFreq
  - duration
  - bandwidth

- **Categorical label**

22 classes of glitches

'Extremely_Loud’	'Repeating_Blips’	'1080Lines’
'Wandering_Line’	'Koi_Fish’	'Low_Frequency_Burst’
'Whistle’	'Scratchy’	'Light_Modulation’
'Blip’	‘Scattered_Light’	'Violin_Mode’
'Power_Line’	'Helix’	'Low_Frequency_Lines’,
'None_of_the_Above’	'1400Ripples’	'Chirp’
'No_Glitch’	'Paired_Doves’	'Air_Compressor’
'Tomte'		



```
list_filename="gspy-db-20180813_filtered.csv"
```

```
data_dir = os.path.join(os.path.dirname(os.getcwd()),"data")
data_dir
```

```
'/Users/roberto/git/2019BragaSchool-gwhandson/data'
```

```
gl_df = pd.read_csv(os.path.join(data_dir,list_filename))
gl_df.describe()
```

Index	GPStime	peakFreq	snr	amplitude	centralFreq	duration
81701.000000	8.170100e+04	81701.000000	81701.000000	8.170100e+04	81701.000000	81701.000000
40850.000000	1.164020e+09	600.612925	25.861831	3.486329e-20	1936.617241	1.271748
23585.191509	1.646547e+07	575.187632	169.987708	2.669169e-19	1360.470233	2.141393
0.000000	1.126403e+09	10.059000	7.500000	2.260000e-23	8.319000	0.001000
20425.000000	1.163774e+09	38.897000	8.212000	1.460000e-22	1015.524000	0.164000
40850.000000	1.165438e+09	262.065000	9.363000	1.980000e-22	1526.092000	0.500000
61275.000000	1.168694e+09	1085.830000	12.598000	3.800000e-22	3234.508000	1.625000
81700.000000	1.205493e+09	2046.281000	11499.370000	2.780000e-17	4727.692000	64.500000

EUROPEAN COOPERATION IN SCIENCE AND TECHNOLOGY

Links to resources

Docker images
https://lip-computing.github.io/datascience2019/docker_images.html

For those who experience technical issues with Docker:

- **Notebooks**
  https://cernbox.cern.ch/index.php/s/VSDpUpsavpmZR4A
  Refer to the *notebooks* folder only, the *data* folder is not updated

- **Data**
  https://owncloud.ego-gw.it/index.php/s/nHXFIJrCvAoDWob

Notebooks

- Jupyter notebooks
  - Random Forest in SKLearn
  - Gradient Boosted Trees in XGBoost
  - Neural Networks in Keras

Evaluation strategy

- Fixed split
  - 66.6% training set
  - 33.3% testing set
  - Seed = 7

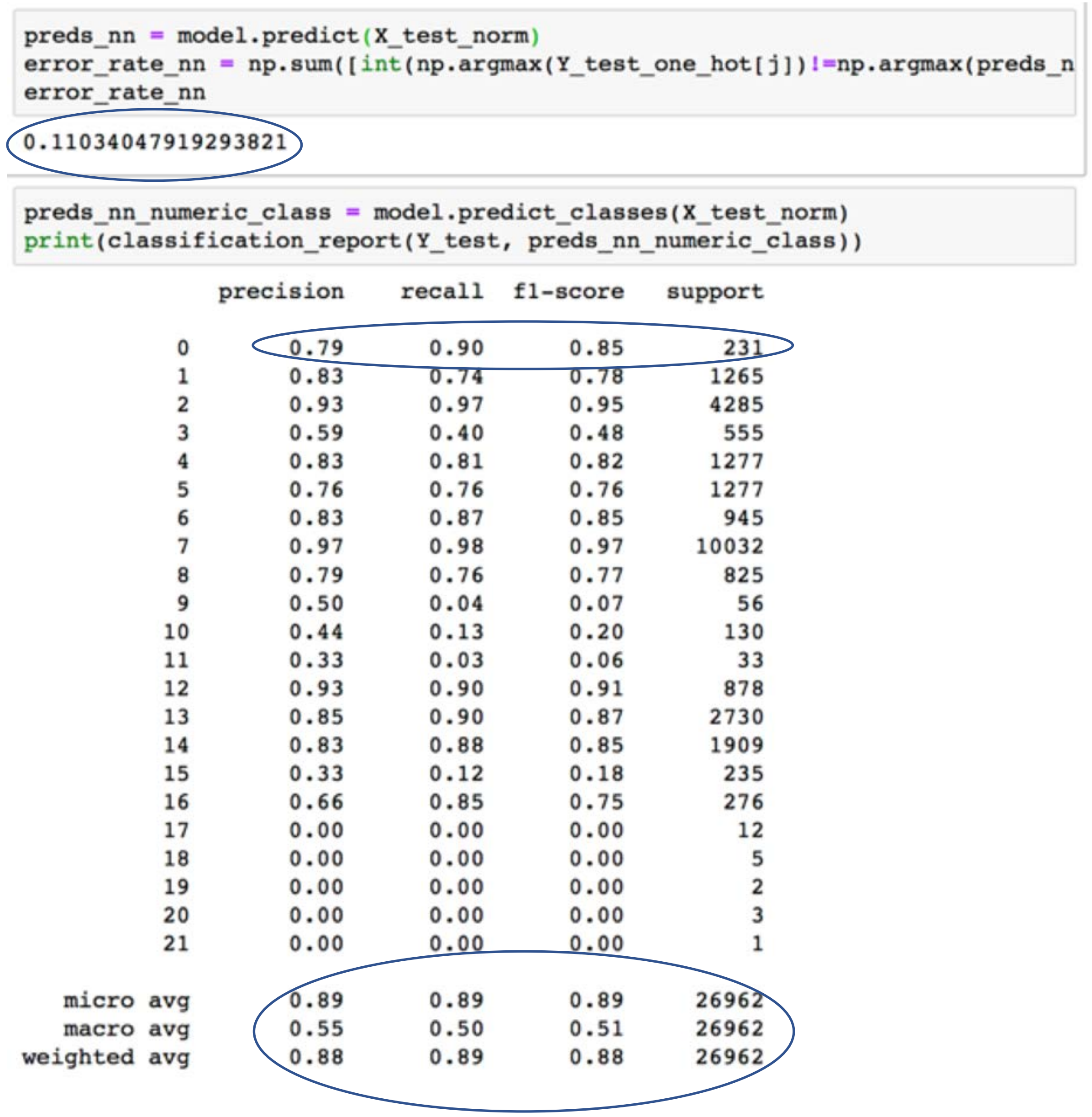
Evaluation metrics

- Error rate
- Precision, Recall, F-Measure (per class)
- Precision, Recall, F-Measure (Micro/Macro/Weigthed)

Task

- Propose a predictive model for glitch classification
- All notebooks include working code to train basic models, and to extract evaluation metrics from predictions
- Identify the best performing model by experimenting with different:
  - Data preprocessing strategies
  - Neural network architectures
  - Grid search over parameter values
  - Stacking different models

Classification accuracy evaluation



```
preds_nn = model.predict(X_test_norm)
error_rate_nn = np.sum([int(np.argmax(Y_test_one_hot[j])!=np.argmax(preds_n
error_rate_nn
```

```
0.11034047919293821
```

```
preds_nn_numeric_class = model.predict_classes(X_test_norm)
print(classification_report(Y_test, preds_nn_numeric_class))
```

```
              precision    recall  f1-score   support

           0       0.79      0.90      0.85       231
           1       0.83      0.74      0.78      1265
           2       0.93      0.97      0.95      4285
           3       0.59      0.40      0.48       555
           4       0.83      0.81      0.82      1277
           5       0.76      0.76      0.76      1277
           6       0.83      0.87      0.85       945
           7       0.97      0.98      0.97     10032
           8       0.79      0.76      0.77       825
           9       0.50      0.04      0.07        56
          10       0.44      0.13      0.20       130
          11       0.33      0.03      0.06        33
          12       0.93      0.90      0.91       878
          13       0.85      0.90      0.87      2730
          14       0.83      0.88      0.85      1909
          15       0.33      0.12      0.18       235
          16       0.66      0.85      0.75       276
          17       0.00      0.00      0.00        12
          18       0.00      0.00      0.00         5
          19       0.00      0.00      0.00         2
          20       0.00      0.00      0.00         3
          21       0.00      0.00      0.00         1

   micro avg       0.89      0.89      0.89     26962
   macro avg       0.55      0.50      0.51     26962
weighted avg       0.88      0.89      0.88     26962
```

Evaluation metrics: Micro vs Macro average

Example

• Class A:	1 TP and 1 FP
• Class B:	10 TP and 90 FP
• Class C:	1 TP and 1 FP
• Class D:	1 TP and 1 FP

$$Pr = \frac{TP}{(TP+FP)}$$

$$Pr_A = Pr_C = Pr_D = 0.5$$

$$Pr_B = 0.1$$

Macro-average $$Pr = \frac{0.5+0.1+0.5+0.5}{4} = 0.4$$

Micro-average $$Pr = \frac{1+10+1+1}{2+100+2+2} = 0.123$$

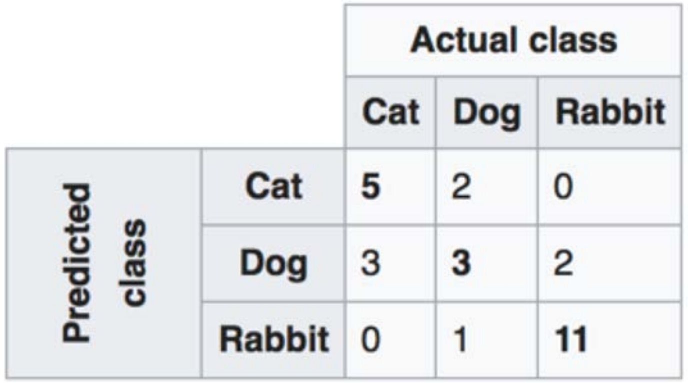
G2NET

COST EUROPEAN COOPERATION IN SCIENCE AND TECHNOLOGY

Evaluation metrics: Confusion matrix

A classification algorithm has been trained to distinguish between cats, dogs and rabbits

Assuming a sample of 27 animals — 8 cats, 6 dogs, and 13 rabbits, the resulting confusion matrix could look like this table



		Actual class		
		Cat	**Dog**	**Rabbit**
Predicted class	**Cat**	**5**	2	0
	Dog	3	**3**	2
	Rabbit	0	1	**11**

- In this confusion matrix, of the 8 actual cats, the algorithm predicted that three were dogs, and of the six dogs, it predicted that one was a rabbit and two were cats.

- We can see from the matrix that the algorithm has trouble distinguishing between cats and dogs, but can make the distinction between rabbits and other types of animals pretty well.

- All correct predictions are located in the diagonal of the table (highlighted in bold), so it is easy to visually inspect the table for prediction errors, as they will be represented by values outside the diagonal.

Confusion matrix in Python:

https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html#sphx-glr-auto-examples-model-selection-plot-confusion-matrix-py

Hyperparameters and Grid Search

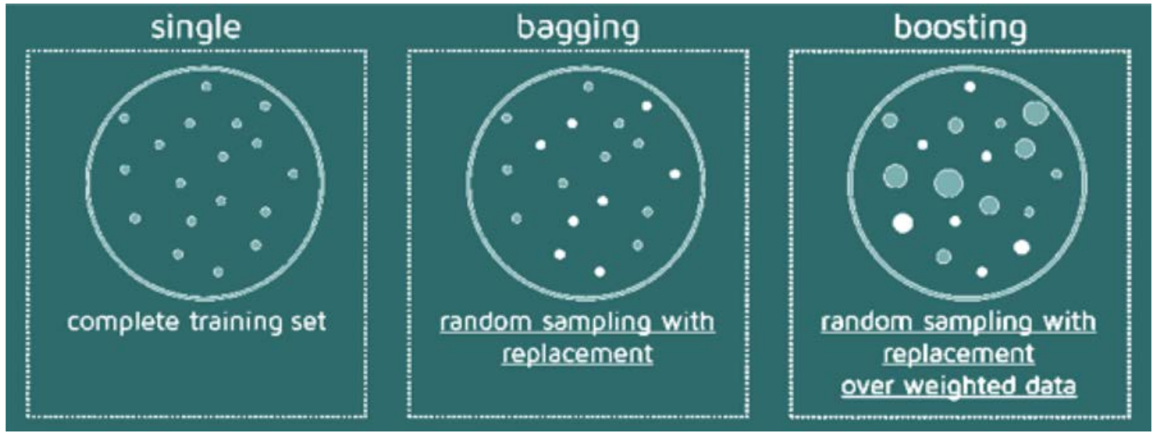
- A model **hyperparameter** is a characteristic of a model that is external to the model and whose value cannot be estimated from data. The value of the hyperparameter has to be set before the learning process begins.
- For example, ***c*** in Support Vector Machines, ***k*** in k-Nearest Neighbors, the ***number of hidden layers*** in Neural Networks.
- Grid-search is used to find the optimal hyperparameters of a model which results in the most 'accurate' predictions.

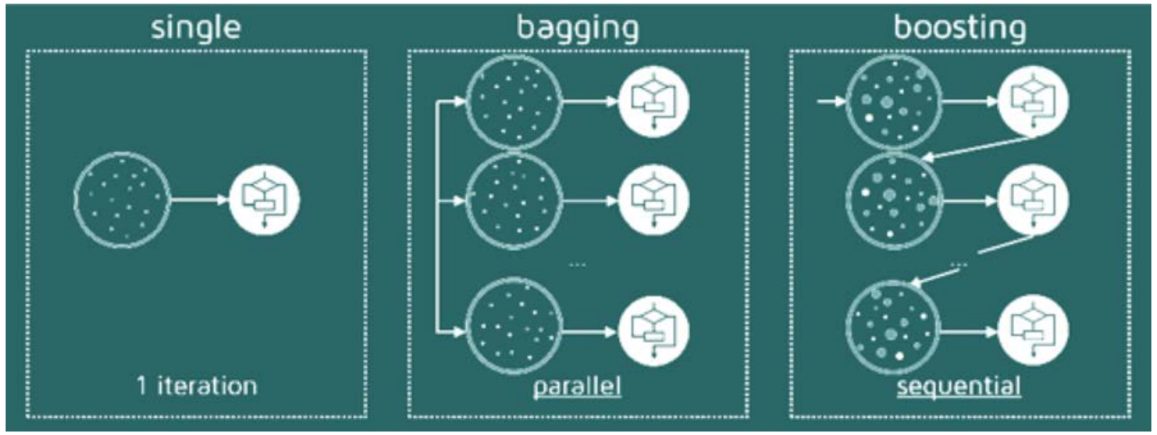
```
from sklearn.model_selection import GridSearchCV
clf = LogisticRegression()
grid_values = {'penalty': ['l1', 'l2'],'C':[0.001,.009,0.01,.09,1,5,10,25]}
grid_clf_acc = GridSearchCV(clf, param_grid = grid_values,scoring = 'recall')

grid_clf_acc.fit(X_train, y_train)
y_pred_acc = grid_clf_acc.predict(X_test)

print('Recall Score : ' + str(recall_score(y_test,y_pred_acc)))
confusion_matrix(y_test,y_pred_acc)
```

Ensemble models
Bagging, boosting, stacking

Ensemble Models: Bagging vs Boosting

- **Bagging** and **Boosting** train multiple learners generating new training data sets by **random sampling with replacement** from the original set.
- In **Bagging**, any element has the same probability to appear in a new data set, and the the training stage is **parallel** (i.e., each model is built independently)
- In **Boosting** the observations are weighted and some of them will take part in the new sets more often. The new learner is learned in a **sequential** way:
    - Each classifier is trained on data, taking into account the previous classifiers' success
    - After each training step, **misclassified data increases its weights** to emphasize the most difficult cases. In this way, subsequent learners will focus on them during their training.


single
bagging
boosting
complete training set
random sampling with replacement
random sampling with replacement over weighted data



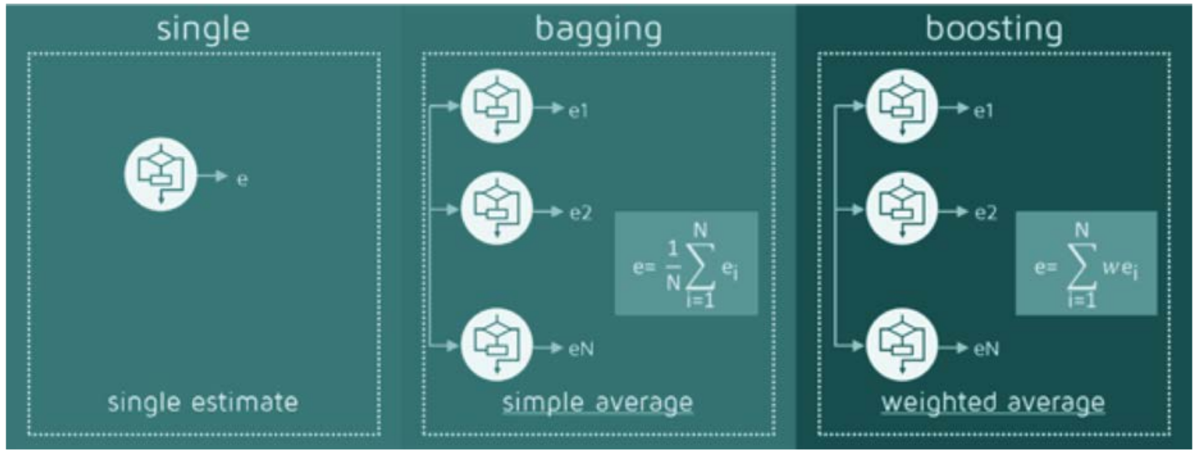
https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/

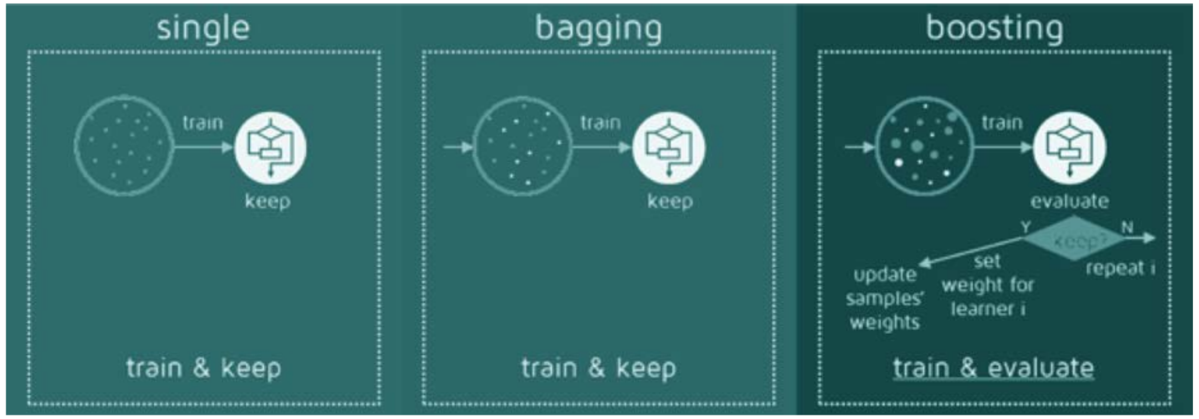
Ensemble Models: Bagging vs Boosting

- To **predict** the class of new data *N* learners are applied to the new observations.
- In **Bagging** the result is obtained by averaging the responses of the *N* learners (or majority vote).

- **Boosting** assigns a second set of weights, this time for the *N* classifiers, in order to take a weighted average of their estimates.
  - During training, the algorithm allocates **weights** to each resulting model. A learner with good a classification result will be assigned a higher weight than a poor one.

  - Boosting techniques may include an **extra-condition** to keep or discard a single learner. In AdaBoost, an error less than 50% is required to maintain the model; otherwise, the iteration is repeated until achieving a learner better than a random guess.



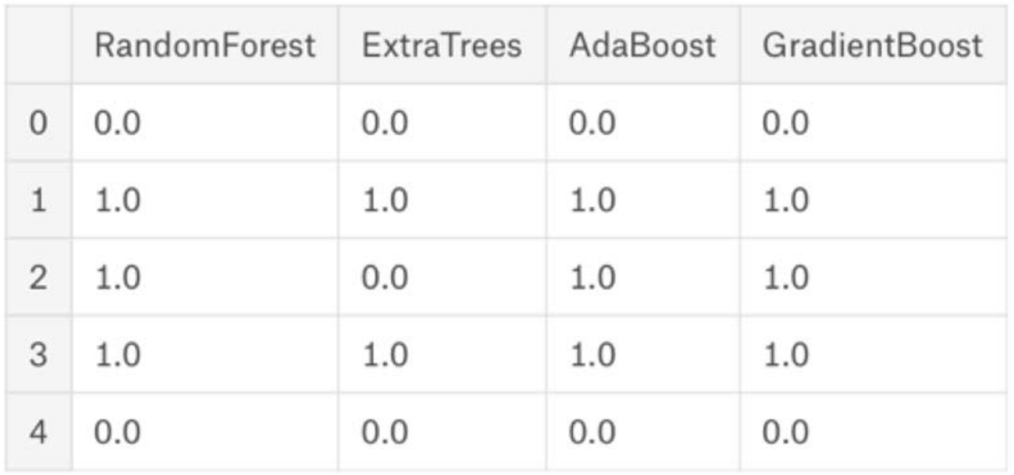


Ensemble Models: Stacking

- **Stacking** uses the predictions of different basic classifiers as a first-level, and then uses another model at the second-level to predict the output from the earlier first-level predictions.

- **Key idea**: predictions of different classifiers can be used as training data for another classifier.

- Stacking generally results in better predictions when the first-level classifiers outcome appear uncorrelated with respect to the specific dataset.



	RandomForest	ExtraTrees	AdaBoost	GradientBoost
0	0.0	0.0	0.0	0.0
1	1.0	1.0	1.0	1.0
2	1.0	0.0	1.0	1.0
3	1.0	1.0	1.0	1.0
4	0.0	0.0	0.0	0.0

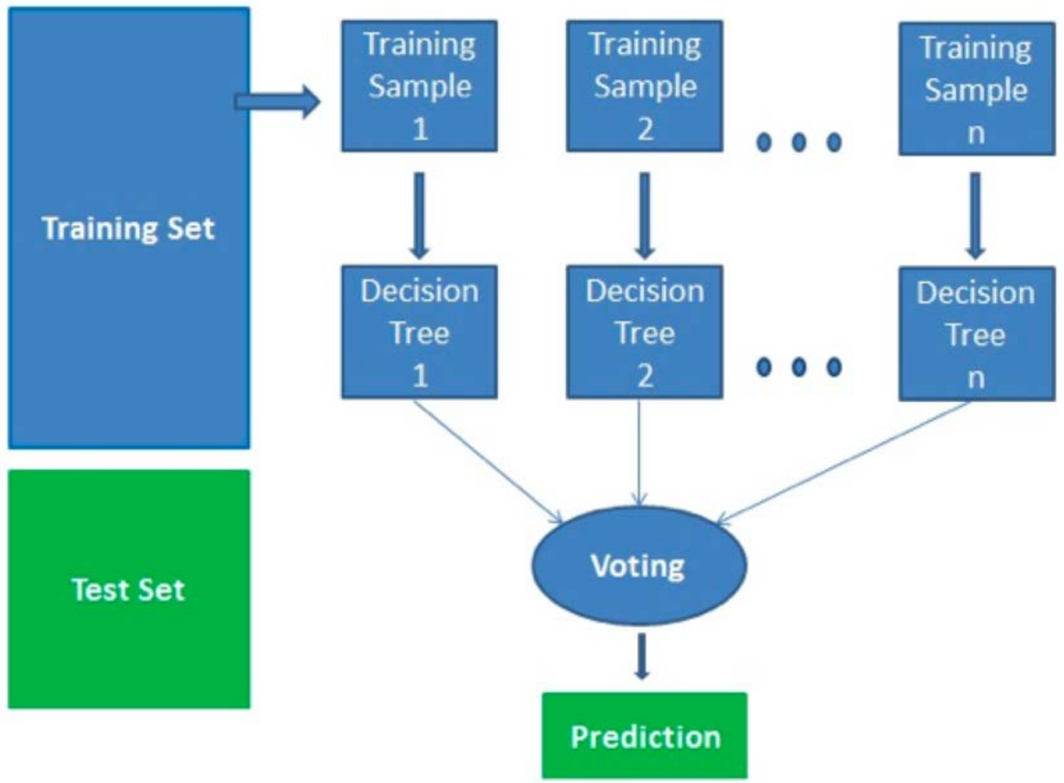
```
base_predictions_train =
pd.DataFrame(
{'RandomForest': rf_preds,
   'ExtraTrees': et_preds,
   'AdaBoost': ada_preds,
   'GradientBoost': gb_preds })
```

https://www.kaggle.com/arthurtok/introduction-to-ensembling-stacking-in-python

Random Forest algorithm

- Random forests create decision trees on randomly selected data samples, gets prediction from each tree and selects the best solution by means of voting.

  1. Select random samples from a given dataset.
  2. Construct a decision tree for each sample and get a prediction result from each decision tree.
  3. Perform a vote for each predicted result.
  4. Select the prediction result with the most votes as the final prediction.



https://www.datacamp.com/community/tutorials/random-forests-classifier-python

- **Key advantages**
  - Highly accurate and robust due to the number of decision trees participating in the process.
  - Does not suffer from overfitting
  - Can be used in both classification and regression problems.
  - Can handle missing values using median values to replace continuous variables, and computing the proximity-weighted average of missing values.

Random Forest in Python sklearn

Most important parameters

• ***n_estimators***	The number of trees in the forest.	default=10
• ***criterion***	Function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain.	
• ***max_depth***	Maximum depth of the tree If *None*, then nodes are expanded until all leaves are pure or until all leaves contain less than ***min_samples_split*** samples	default=None
• ***min_samples_split***	Minimum number of samples to split an internal node Can be an integer value or a float representing a fraction, such that *ceil(**min_samples_split** * **n_samples**)* are the minimum number of samples for each split	default=2
• ***min_samples_leaf***	The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least ***min_samples_leaf*** training samples in each of the left and right branches.	default=1

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Random Forest in Python sklearn

Jupyter notebook

- *RandomForest.ipynb*

Required code fixes

Setup correct dataset filename and path:

```
list_filename="gspy-db-20180813_O1_filtered_t1126400691-1205493119_snr7.5_tr_gspy.csv”
data_dir = os.path.join(os.path.dirname(os.getcwd()),"data")
```

Update attribute list:

```
X = gl_df.get(['GPStime','peakFreq','snr','centralFreq','duration','bandwidth'])
```

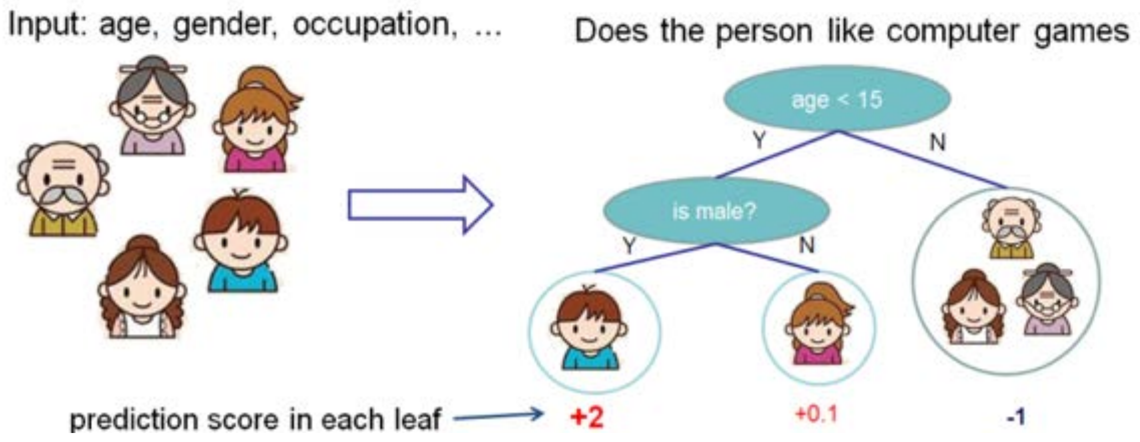
XGBoost: Core Model

- Based on the **tree ensemble model:** it consists of a set of classification and regression trees (CART)
- It sums the prediction of multiple trees together, which try to complement each other

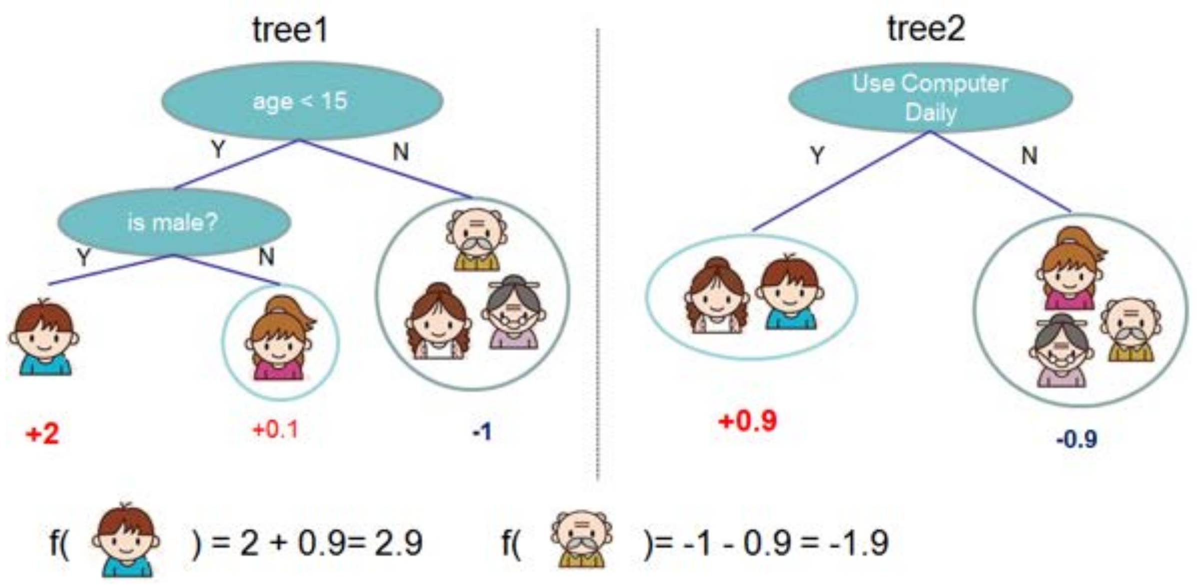
$$\hat{y_i} = \sum_{k=1}^{K} f_k(x_i), f_k \in \mathcal{F}$$

K is the number of trees, f is a function in the functional space $\mathcal{F}$, and $\mathcal{F}$ is the set of all possible CARTs.

Single tree example


Input: age, gender, occupation, ...
Does the person like computer games
age < 15
Y
N
is male?
Y
N
prediction score in each leaf
+2
+0.1
-1

Multiple trees example



XGBoost: Tree Boosting

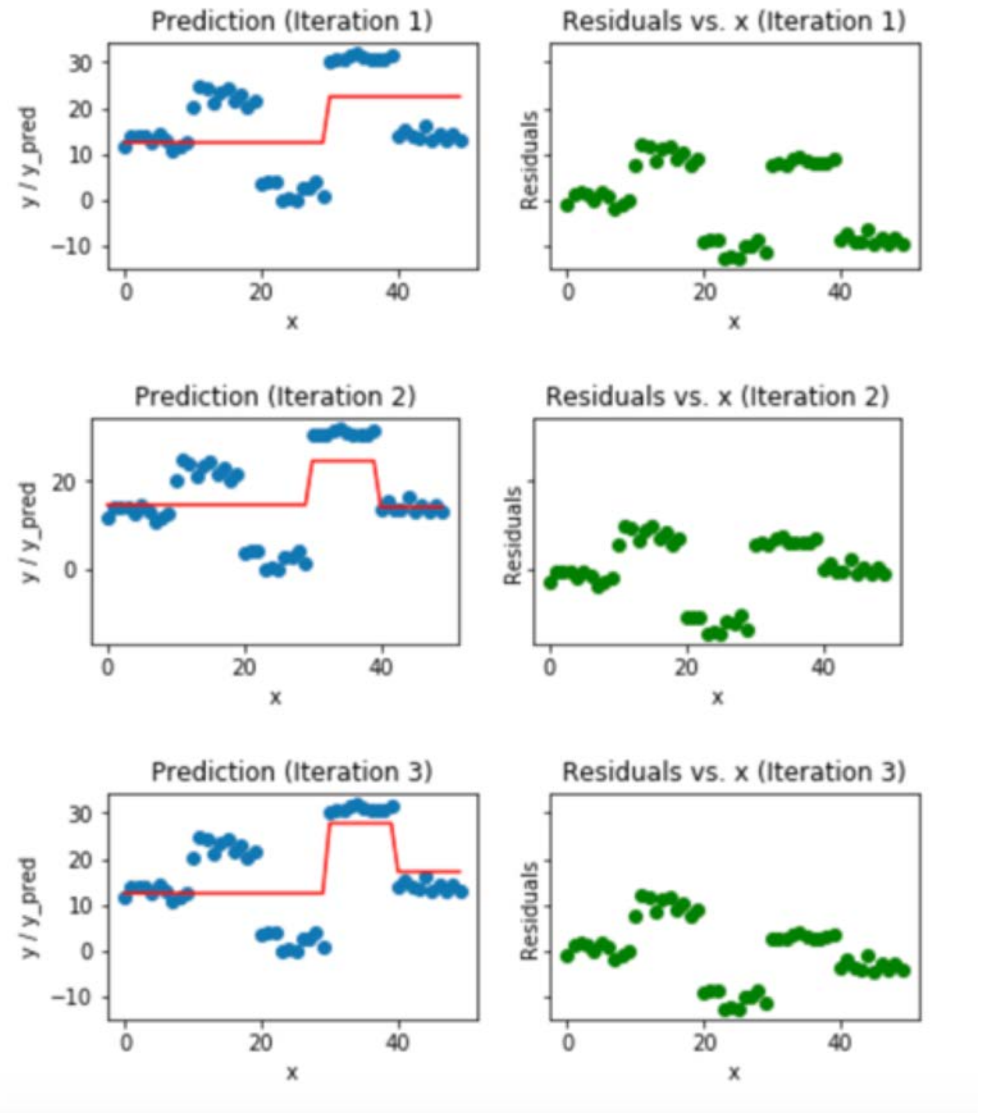
- Trees are learned by defining and optimizing an **objective function:**

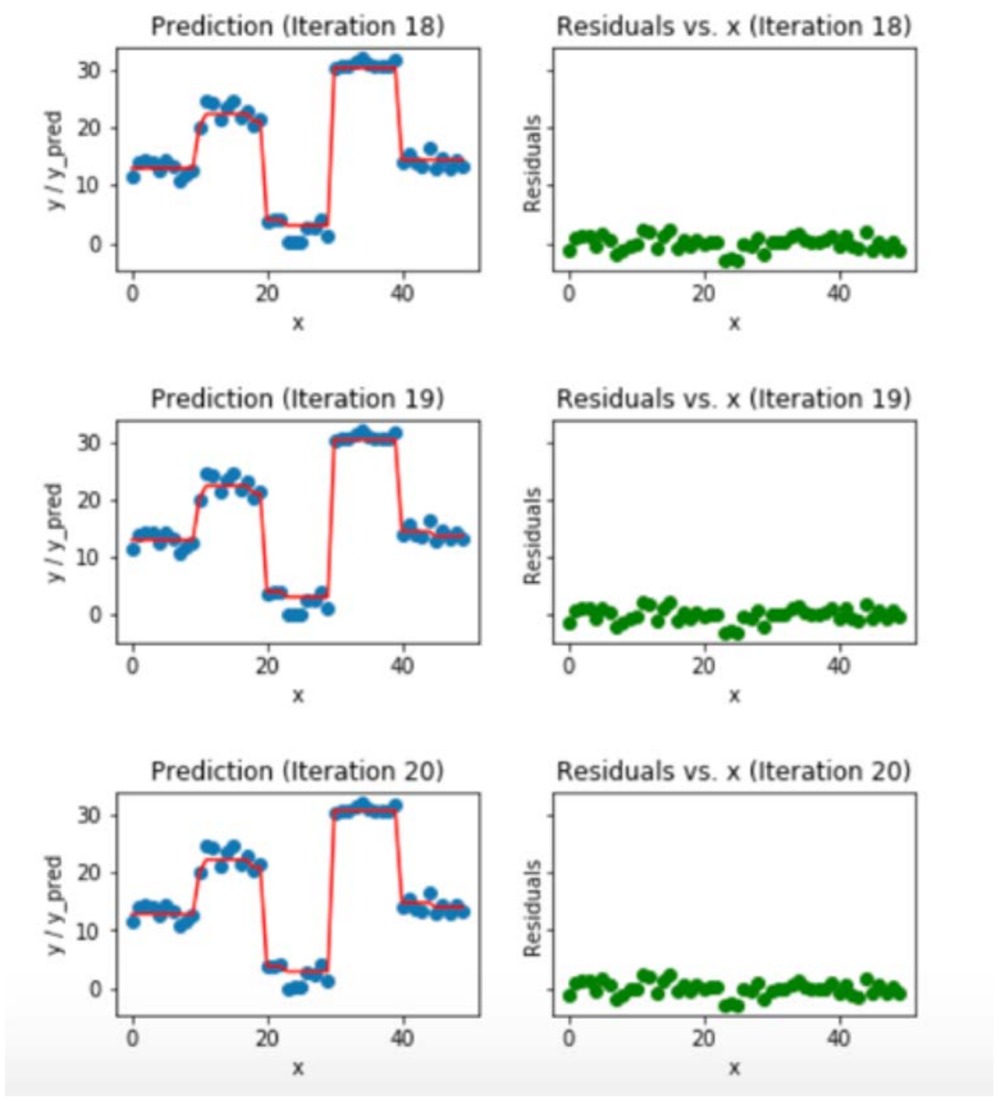
$$\text{obj}(\theta) = \sum_{i}^{n} l(y_i, \hat{y_i}) + \sum_{k=1}^{K} \Omega(f_k)$$

$$\begin{aligned}
\hat{y}_i^{(0)} &= 0 \\
\hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\
\hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \\
&\dots \\
\hat{y}_i^{(t)} &= \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)
\end{aligned}$$

EUROPEAN COOPERATION IN SCIENCE AND TECHNOLOGY

Boosted Trees Performance





https://medium.com/mlreview/gradient-boosting-from-scratch-1e317ae4587d

XGBoost: Data format

- It currently supports two text formats for ingesting data: LibSVM and CSV.

- You may specify instance weights in the LibSVM file by appending each instance label with the corresponding weight in the form of **[label]:[weight]**

- Supports **numeric data only**. Categorical features can be processed transforming them:

  - To **numeric features** (using **`LabelEncoder`**)
    - Example: [a,b,b,c] > array([0, 1, 1, 2])

  - To **binary features** with One-Hot-Encoding (using **`OneHotEncoder`**)
    - Example: [a,b,b,c] > array([[1., 0., 0.], [0., 1., 0.], [0., 1., 0.], [0., 0., 1.]])

  This is the ideal representation of a categorical variable for XGBoost or any other machine learning algorithm.

https://xgboost.readthedocs.io/

G2NET

cost EUROPEAN COOPERATION IN SCIENCE AND TECHNOLOGY

XGBoost: Example code

- Example code:

```
training = xgb.DMatrix(X_train, label=Y_train)
test = xgb.DMatrix(X_test, label=Y_test)

num_round = 2
param = {'max_depth': 2,
            'eta': 1,
            'silent':1,
            'objective':'multi:softmax',
            'num_class': Y_count_labels}
bst = xgb.train(param, training, num_round)
preds = bst.predict(test)
```

DMatrix is a internal data structure that used by XGBoost which is optimized for both memory efficiency and training speed. You can construct DMatrix from **numpy.arrays**

Example featured in the XGBoost Jupyter Notebook for the glitch classification data challenge

XGBoost: Control overfitting and imbalance

- When you observe **high training accuracy**, but **low test accuracy**, it is likely to be an overfitting problem.

- There are in general two ways that you can control overfitting in XGBoost:

  - The first way is to directly control model complexity
    This includes **`max_depth`**, **`min_child_weight`** and **`gamma`**

  - The second way is to add randomness to make training robust to noise
    This includes **`subsample`** and **`colsample_bytree`**

  - You can also reduce stepsize **`eta`**. Remember to increase **`num_round`** when you do so.

- In some cases, the dataset is also extremely imbalanced. This can affect the training of XGBoost model. For classification, you can balance the positive and negative weights via **`scale_pos_weight`** (default 1)
- A typical value to consider: **`sum(negative instances)/sum(positive instances)`**

Gradient Boosted Trees in XGBoost

Jupyter notebook

- *XGBoost.ipynb*

Required code fixes

Setup correct dataset filename and path:

```
list_filename="gspy-db-20180813_O1_filtered_t1126400691-1205493119_snr7.5_tr_gspy.csv”
data_dir = os.path.join(os.path.dirname(os.getcwd()),"data")
```

Update attribute list:

```
X = gl_df.get(['GPStime','peakFreq','snr','centralFreq','duration','bandwidth'])
```

Auto-Encoders

Auto-Encoders

Auto-Encoders learn to reconstruct a given input representation with a low reconstruction error.

A suitable way to learn an auto-encoder consists in layer-wise back-propagation learning.

Each auto-encoder has an encoding function **γ** and a decoding function **δ** such that:

$$\gamma : \mathcal{X} \to \mathcal{F}, \qquad \delta : \mathcal{F} \to \mathcal{X}$$

$$\gamma, \delta = \arg\min_{\gamma,\delta} \|X - \delta(\gamma(X))\|^2$$

Auto-Encoders

Encoding stage (one hidden layer)

Takes the input ***x*∈R^d =X** and maps it to an hidden representation ***z*∈R^p =F**

Where **σ** is a sigmoid or a rectified linear unit activation function, **W** is a weight matrix and **b** is a bias vector.

$$\mathbf{z} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

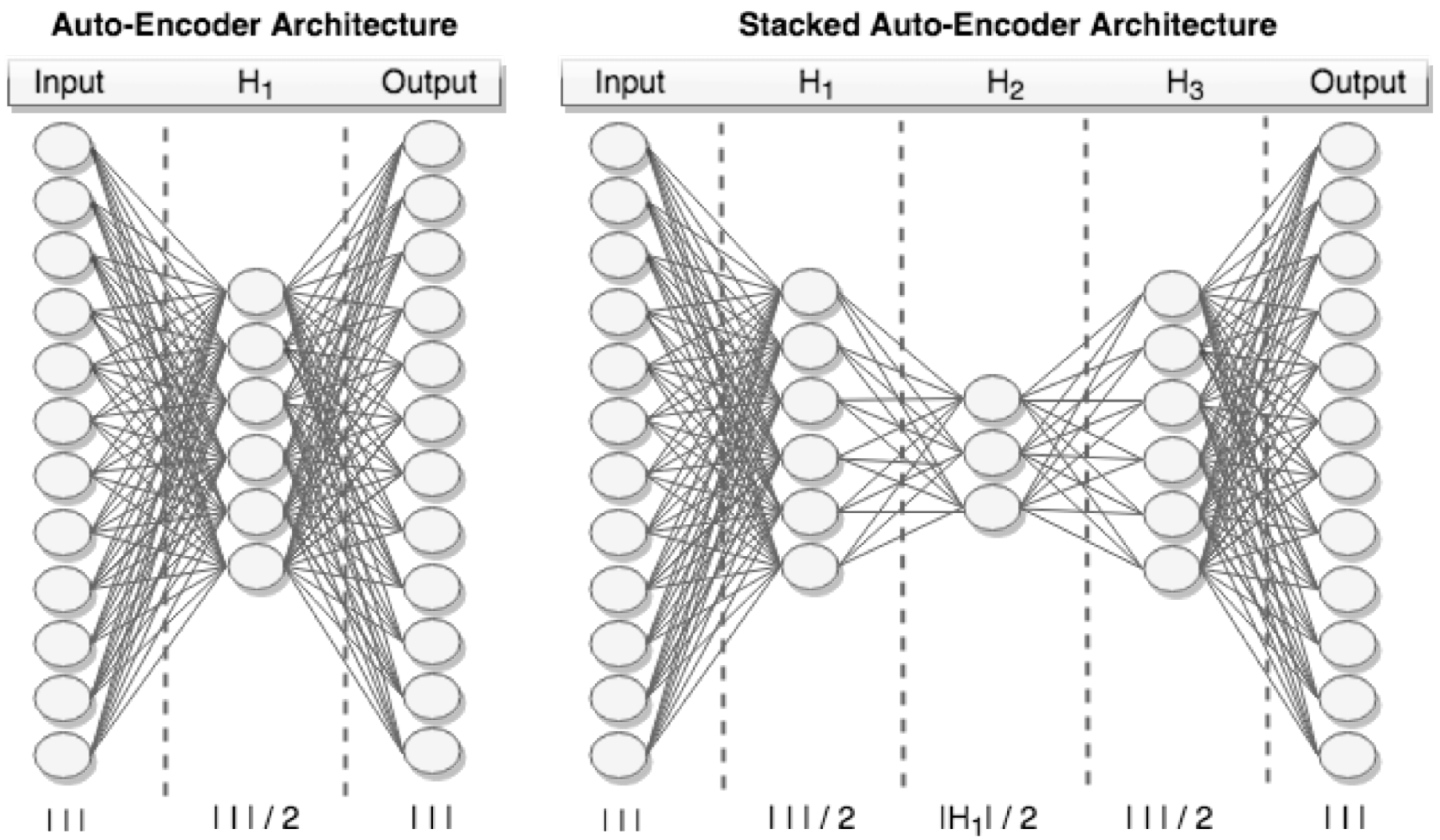
Decoding stage (one hidden layer)

The decoding stage reconstructs *x* from *z* as: $\mathbf{x}' = \sigma'(\mathbf{W}'\mathbf{z} + \mathbf{b})$

such that the following loss is minimized:

$$\mathcal{L}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|^2 = \|\mathbf{x} - \sigma'(\mathbf{W}'(\sigma(\mathbf{W}\mathbf{x} + \mathbf{b})) + \mathbf{b}')\|^2$$

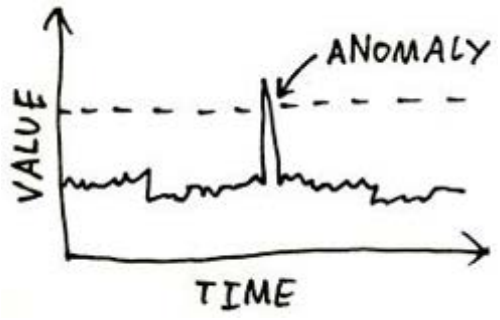
Auto-Encoders vs Stacked Auto-Encoders



Auto-Encoders: Possible tasks

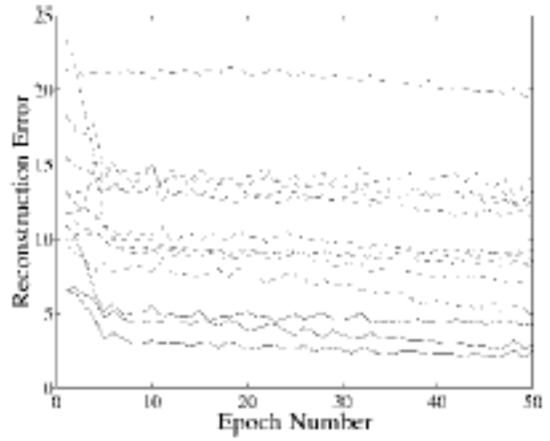
Anomaly detection

- Once the auto-encoder is trained with non-anomalous data, a high reconstruction error for a new instance means that it is possibly an anomaly.



Clustering

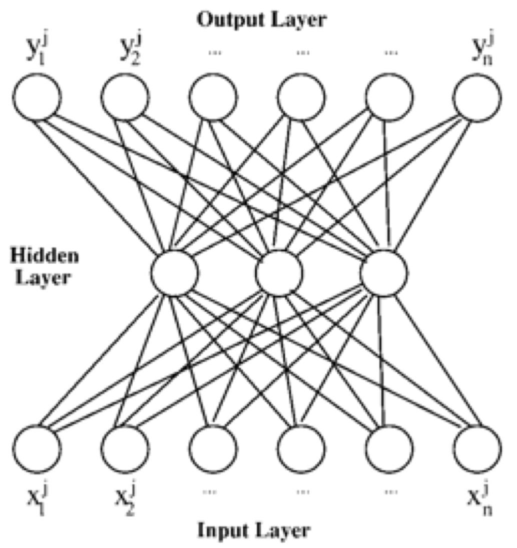
- Non-linear auto-encoders build multiple-local-valley representations of the underlying domain.
- Instances with similar values of reconstruction error may imply that they belong to the same cluster.



Auto-Encoders: Possible tasks

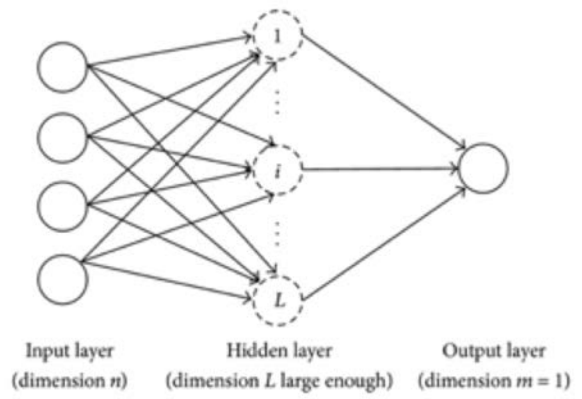
Recognition-based classification

- Once trained with data belonging to the positive class, if the **reconstruction error** is lower than a **threshold** for an unseen example, it belongs to the positive class, otherwise it belongs to the negative class.



Concept learning prior to classification or regression

- Perform **layer-wise pre-training**
- Trained layers can be copied to other neural network models (a new model with one output neuron for classification)
- Pre-training should initialize the weights closer to good solutions (see Larochelle et al. 2009)

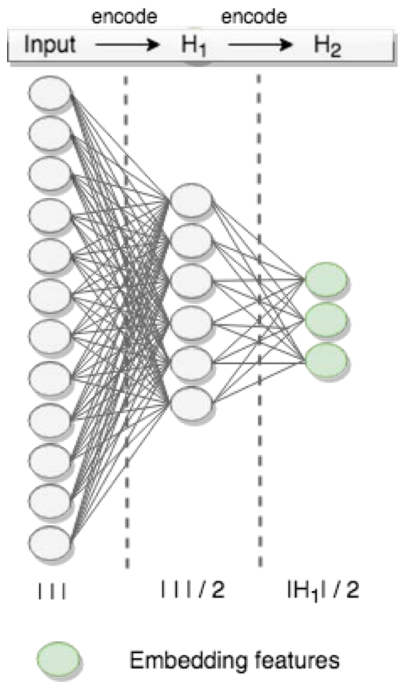

1
i
L
Input layer (dimension n)
Hidden layer (dimension L large enough)
Output layer (dimension m = 1)

Auto-Encoders: Possible tasks

Feature extraction

- After training, extract a set of features of reduced dimensionality (**embedding features**) exploiting the encoding function.

- Reduced dimensionality implies model compactness and possible mitigation of collinearity effects, similarly to Principal Component Analysis (PCA).

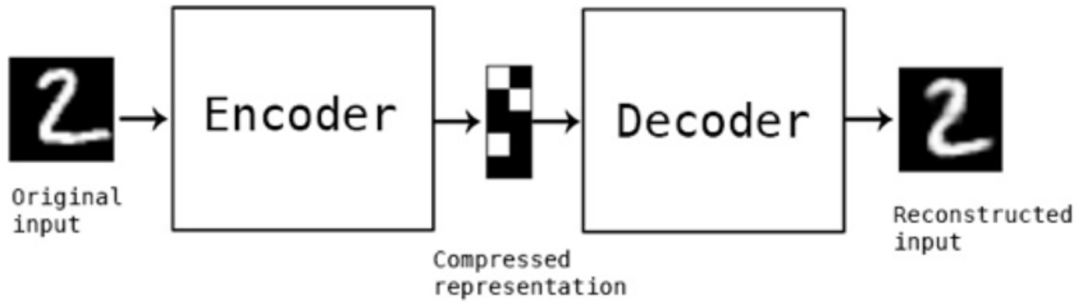


Note: Auto-encoder embedding features are equivalent to PCA just if the hidden layer has linear activations (see Japkowicz et al. 2000)

Keras: Feature extraction via autoencoders

Auto-encoders

- Data compression algorithm where the compression and decompression functions are learned automatically from examples rather than engineered by a human
- In almost all contexts where the term "autoencoder" is used, the compression and decompression functions are implemented with neural networks
- Applications of autoencoders are **data denoising** and **dimensionality reduction**



```
from keras.layers import Input, Dense
from keras.models import Model

encoding_dim = 32
input = Input(shape=(784,))

encoded = Dense(encoding_dim, activation='relu')(input)
decoded = Dense(784, activation='sigmoid')(encoded)
autoencoder = Model(input, decoded)

encoder = Model(input_img, encoded)
```

https://blog.keras.io/building-autoencoders-in-keras.html

Keras: Feature extraction

Deep auto-encoders

- Instead of a single layer as encoder or decoder, they implement a stack of layers

```
encoded = Dense(128, activation='relu')(input_img)
encoded = Dense(64, activation='relu')(encoded)
encoded = Dense(32, activation='relu')(encoded)

decoded = Dense(64, activation='relu')(encoded)
decoded = Dense(128, activation='relu')(decoded)
decoded = Dense(784, activation='sigmoid')(decoded)

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adadelta',
loss='binary_crossentropy')

autoencoder.fit(x_train, x_train,
                epochs=100,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
```

https://blog.keras.io/building-autoencoders-in-keras.html

Neural Networks in Keras: Classification example

Jupyter notebook

- *KerasNN.ipynb*

Required code fixes

Setup correct dataset filename and path:

```
list_filename="gspy-db-20180813_O1_filtered_t1126400691-1205493119_snr7.5_tr_gspy.csv”
data_dir = os.path.join(os.path.dirname(os.getcwd()),"data")
```

Update attribute list:

```
X = gl_df.get(['GPStime','peakFreq','snr','centralFreq','duration','bandwidth’])
```

```
Setup correct number of neurons in the input layer:
model = Sequential()
model.add(Dense(70, input_dim=6, activation='tanh'))
model.add(Dense(22, activation='softmax'))
```