Selecting & improving ML models

Lecture outline

- Model selection and improving
  - Hyperparameter tuning
  - Cross validation
- Challenges
  - Bias and variance
  - Class imbalance
  - Ethics
- Improving ML models with ensemble methods

Improving models with hyperparameter tuning

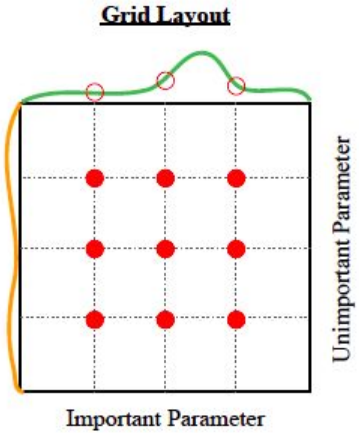
Parameters vs Hyperparameters

- Parameters (or weights) of a model are learned from the data, typically in an iterative way, typically by optimizing a training objective
  - E.g. weights of a linear regression model, weights of a neural network,...
- Hyperparameters control how those parameters are learned
  - E.g. K in KNN, depth of decision trees, learning rate in a neural network,...

Ways of optimizing/tuning hyperparameters

1. Manual tuning
2. **Grid search**
3. **Random search**
4. Bayesian optimization
5. Population-based training (PBT)

Grid search



- Try every possible combination of each set of hyper-parameters.
- It’s easy to implement and can be parallelized
- It’s an exhaustive search that might only be suitable for a small search space. It doesn’t scale well

Grid search (2)

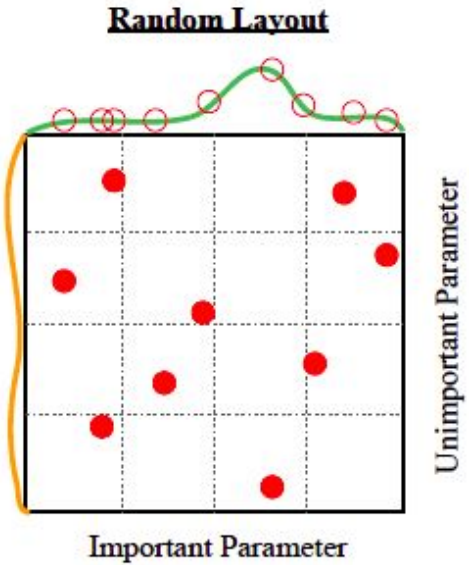
```
parameters ={'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100],
     'criterion' : ['gini', 'entropy'],
     'max_features': [0.3,0.5,0.7,0.9],
     'min_samples_leaf': [3,5,7,10,15],
     'min_samples_split': [2,5,10],
     'n_estimators': [50,100,200,400,600]}


from sklearn.model_selection import ParameterGrid
param_size = ParameterGrid(parameters)
len(param_size)


Output:
12000
```

The grid size will be 10*2*4*5*3*5 = 12000

Output:

```
Best Params:  {'criterion': 'gini', 'max_depth': 90, 'max_features': 'log2', 'min_samples_leaf': 5, 'min_s
amples_split': 10, 'n_estimators': 50}
Best Score:  0.8412587412587413
CPU times: user 3min 49s, sys: 6.52 s, total: 3min 56s
Wall time: 4h 49min 25s
```

Randomized search



- Instead of trying every possible combination, this chooses the hyperparameter sample combinations randomly from grid space
- There is no guarantee that we will find the best result like Grid Search
- Can be extremely effective in practice as computational time is very less

Randomized search (2)

```
%%time
from sklearn.model_selection import RandomizedSearchCV
random_search=RandomizedSearchCV(estimator = RandomForestClassifier(),
param_distributions=parameters,verbose=1, n_jobs=-1,
                                 n_iter=200)
random_result = random_search.fit(X_train, y_train)
print('Best Score: ', random_result.best_score_*100)
print('Best Params: ', random_result.best_params_)
```

Output

```
Best Score:  83.84221412390428
Best Params:  {'n_estimators': 400, 'min_samples_split': 5, 'min_samples_leaf': 5, 'max_features': 0.3, 'm
ax_depth': 50, 'criterion': 'entropy', 'bootstrap': True}
CPU times: user 5.5 s, sys: 178 ms, total: 5.68 s
Wall time: 5min 20s
```

Useful libraries for hyperparameter tuning

1. Scikit-learn
2. Scikit-Optimize
3. Optuna
4. Hyperopt

Selecting models with cross validation

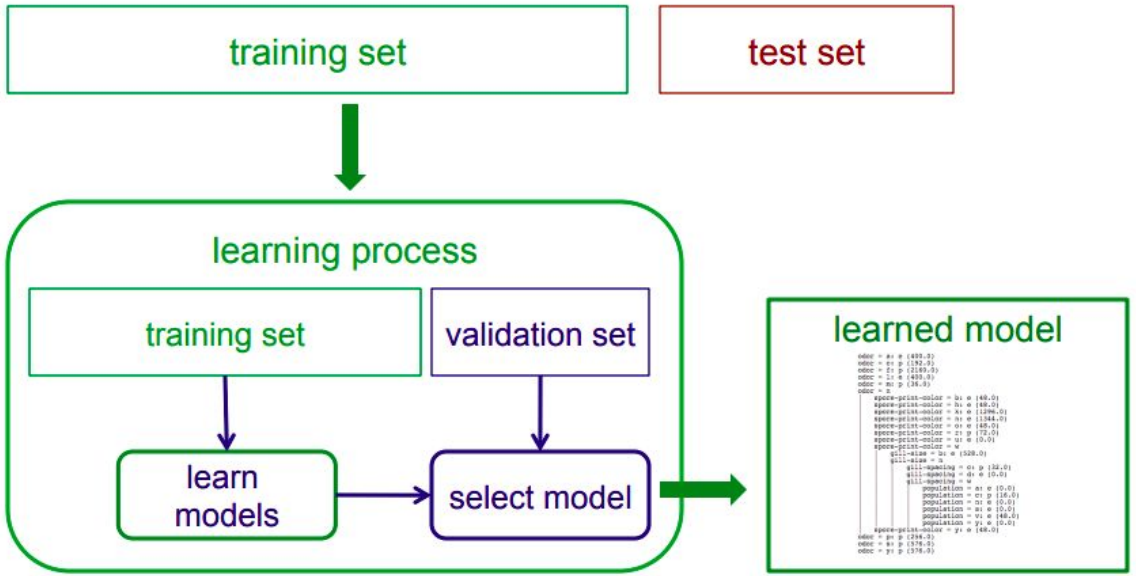
Test set revisited

How can we get an unbiased estimate of the accuracy of a learned model?

- when learning a model, you should pretend that you don’t have the test data yet
- If the test set labels influence the learned model in any way, accuracy estimates will be biased (“data leakage”)

Revisiting validation / tuning / dev sets

Suppose we want unbiased estimates of accuracy during the learning process (e.g. to choose the best level of decision-tree pruning)?

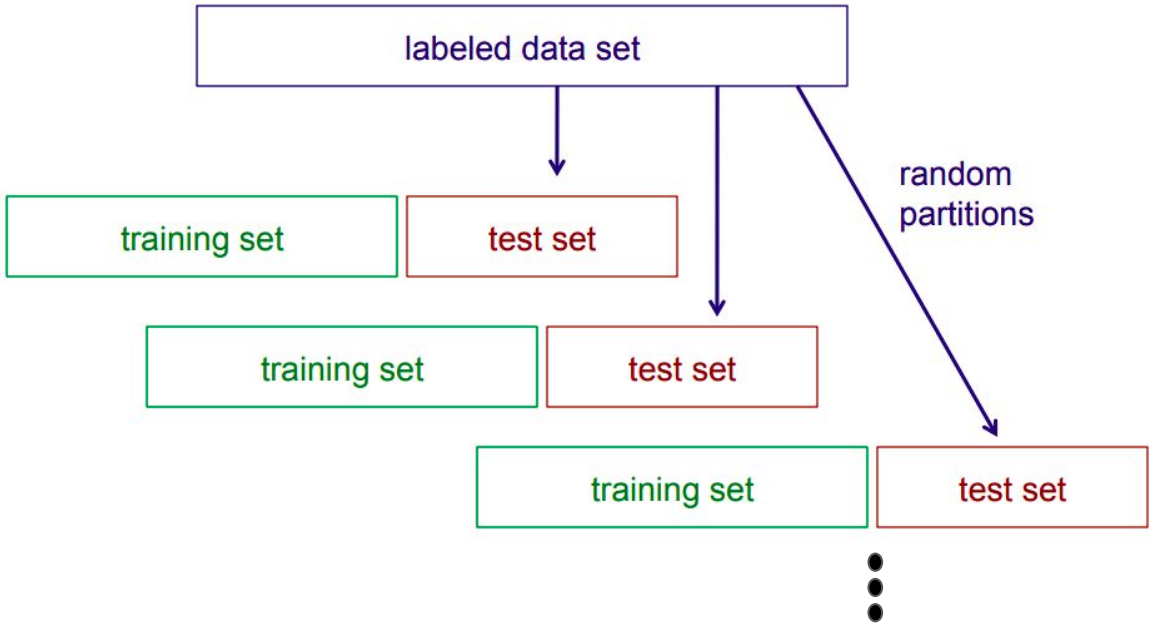


Limitations of using a single training/test partition

- We may not have enough data to make sufficiently large training and test sets
  - A larger **test** set gives us more reliable estimate of the real world accuracy (i.e. a lower variance estimate)
  - but… a larger **training** set will be more representative of the real world process for better learning
- A single training set doesn't tell us how sensitive accuracy is to a particular training sample
  - The accuracy may differ from sample (training set) to sample

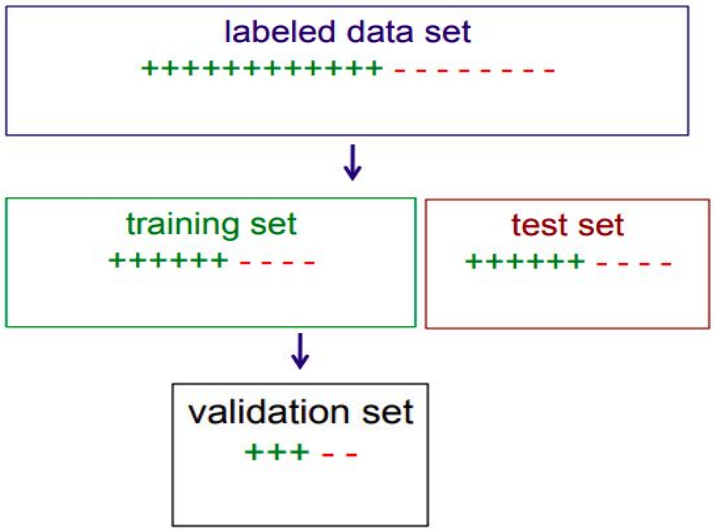
Random resampling

We can address the second issue by repeatedly randomly partitioning the available data into training and set sets.



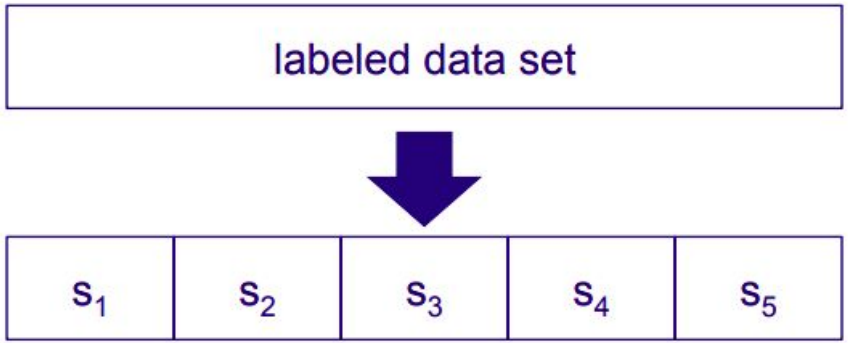
Stratified sampling

When randomly selecting training or validation sets, we may want to ensure that **class proportions are maintained in each selected set**



K fold Cross Validation

- Partition data into n subsamples
- iteratively leave one subsample out for the test set, train on the rest



iteration	train on	test on
1	s_2 s_3 s_4 s_5	s_1
2	s_1 s_3 s_4 s_5	s_2
3	s_1 s_2 s_4 s_5	s_3
4	s_1 s_2 s_3 s_5	s_4
5	s_1 s_2 s_3 s_4	s_5

Cross validation example

Suppose we have 100 instances, and we want to estimate accuracy with cross validation

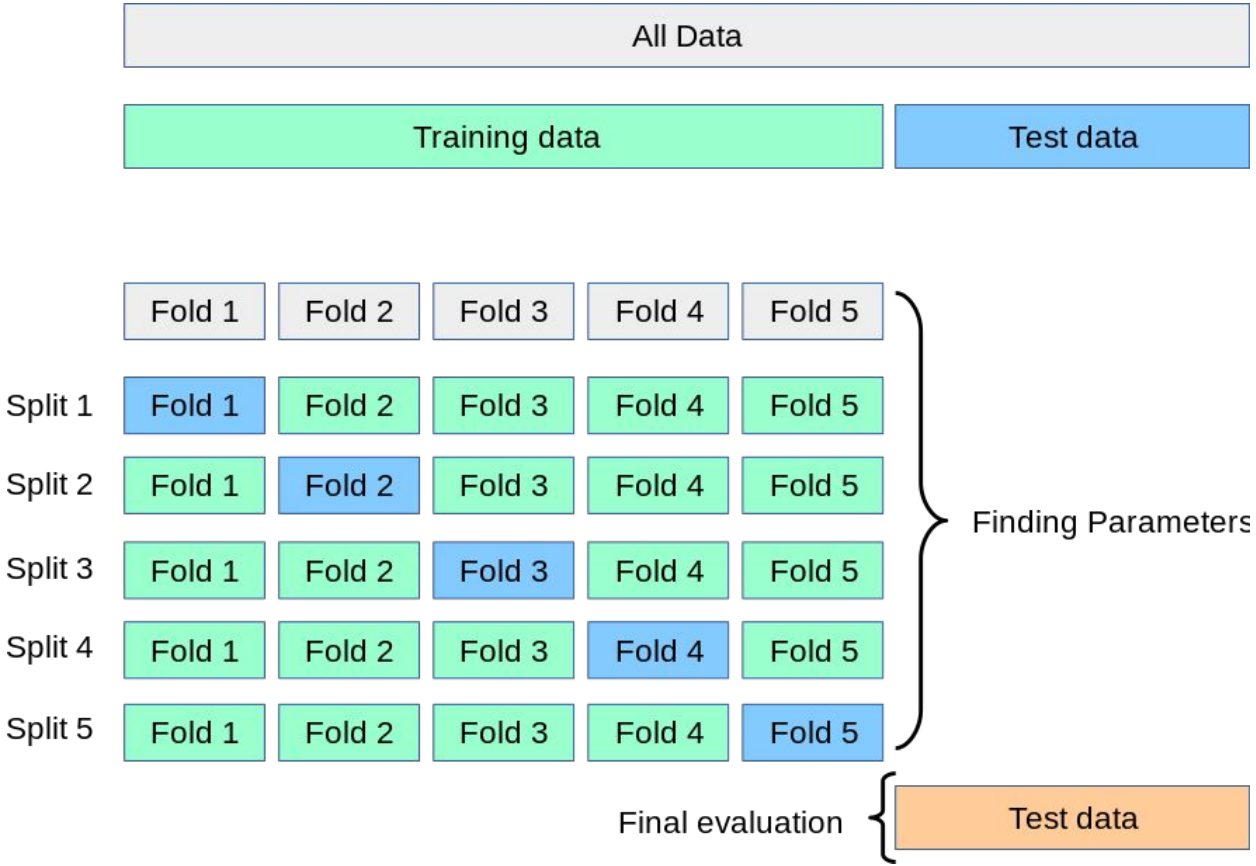
iteration	train on	test on	correct
1	S_2 S_3 S_4 S_5	S_1	11 / 20
2	S_1 S_3 S_4 S_5	S_2	17 / 20
3	S_1 S_2 S_4 S_5	S_3	16 / 20
4	S_1 S_2 S_3 S_5	S_4	13 / 20
5	S_1 S_2 S_3 S_4	S_5	16 / 20

Validation accuracy = 73/100 = 73%

Cross validation

- 10-fold cross validation is common, but smaller values of n are often used when learning takes a lot of time (e.g. 5)
- in **leave-one-out cross validation**, n = # instances
- in **stratified cross validation**, stratified sampling is used when partitioning the data

Summary: Hyperparameter tuning and model selection with cross validation



Reference

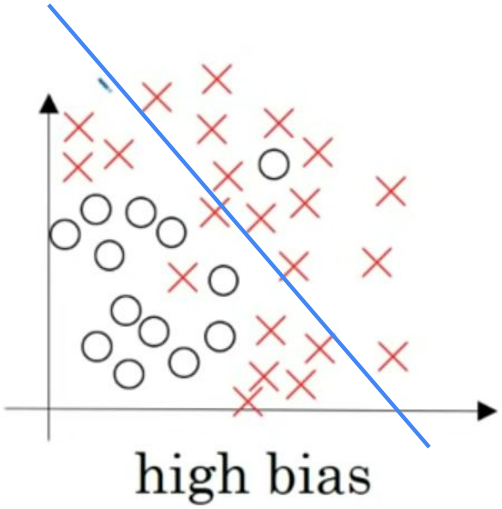
Issues you might face during an ML project

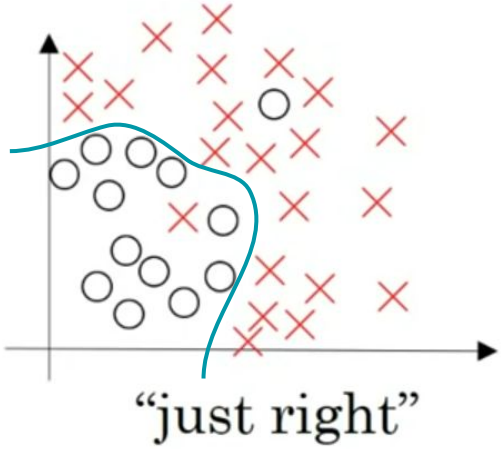
Challenges

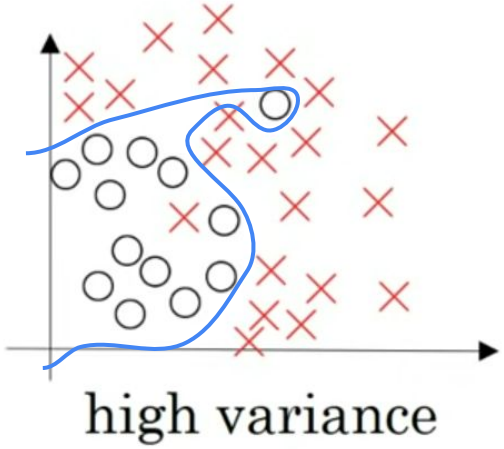
1. Poor quality of data
2. Lack of training data
3. Class imbalance
4. Bias variance tradeoff
5. Resource requirements (hardware)
6. Implementation issues
    a. Library incompatibilities
7. Scalability
8. Interpretability
9. Ethics

Bias and Variance

Bias and Variance



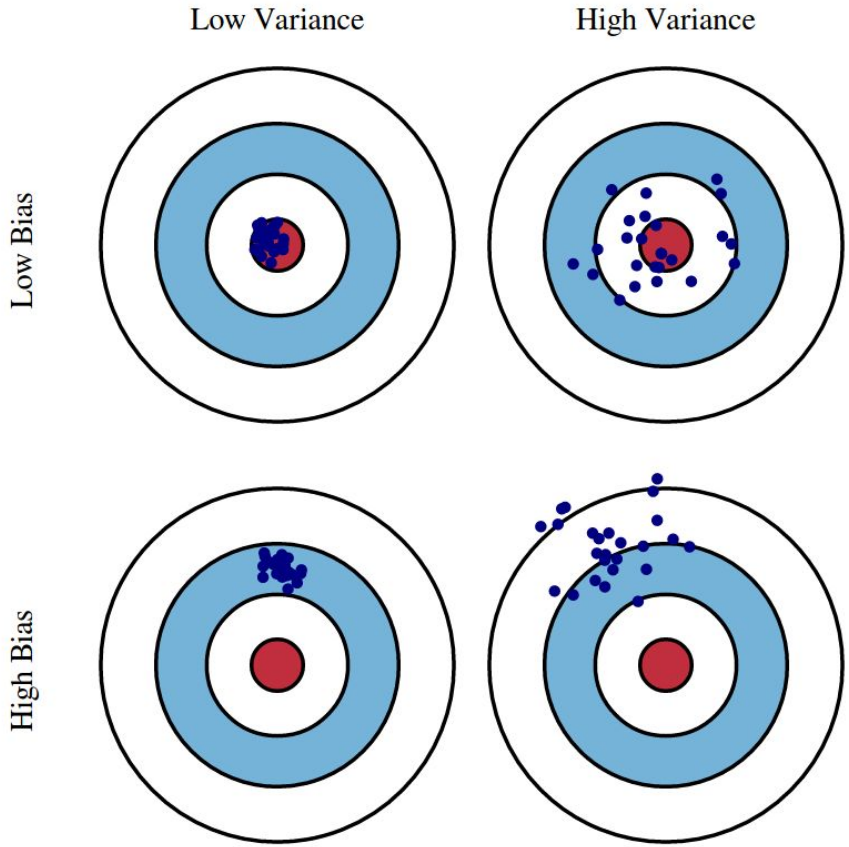




Underfitting

Overfitting

Bullseye Bias Variance Illustration



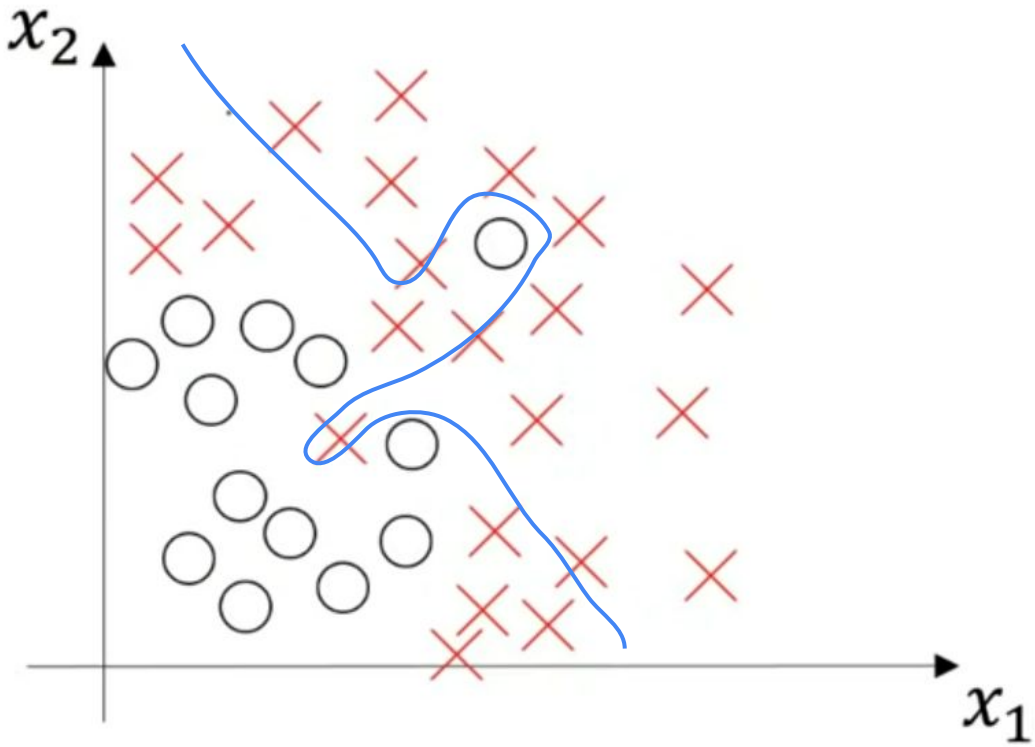
Source: http://scott.fortmann-roe.com/docs/BiasVariance.html

How to spot bias and variance

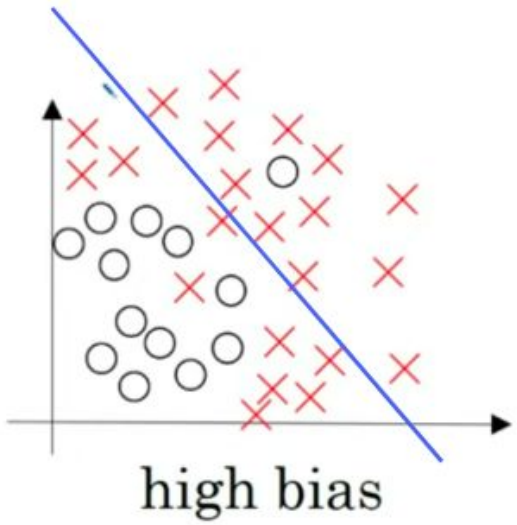
Train set error	2%	15%	15%	0.5%
Dev set error	10%	16%	30%	1%
	High variance	**High bias**	**High bias and high variance**	**Low bias low variance**

The numbers are based on the assumption that the human error (or the best possible error, or the Bayes (optimal) error is close to zero.

High bias and high variance



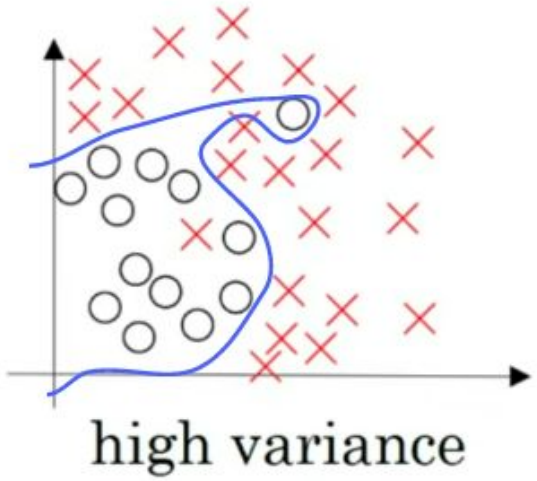
High bias



Underfitting

- Training error is too high
- What to do?
  - Increase model complexity
  - Train for longer if the algorithm allows it
  - Try more advanced optimization methods, if the algorithm allows it
  - Try different models and hyperparameters

High variance



Overfitting

- Error on the dev set is high
- What to do?
  - Get more training data
  - Try regularization
  - Try different models, hyperparameters

Mathematical basis of bias and variance

Assume we want to estimate the mapping f that gives us the value of the dependent variable Y, for a value of X

$$Y=f(X) + e$$

Where e is the error term and its normally distributed with zero mean

Expected squared error at a point x

$$Err(x) = E\left[(Y - \hat{f}\,(x))^2\right]$$

Err(x) can be further decomposed as

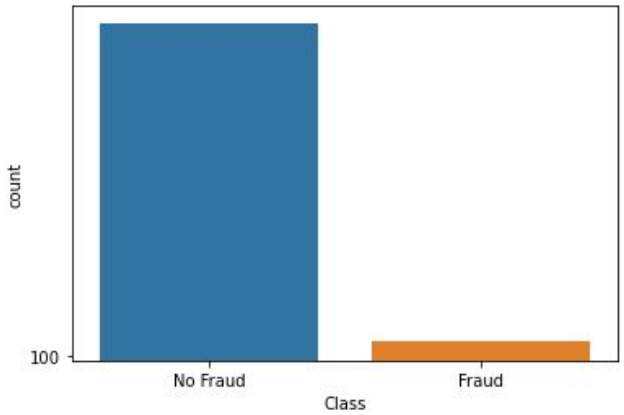
$$Err(x) = \left(E[\hat{f}\,(x)] - f(x)\right)^2 + E\left[\left(\hat{f}\,(x) - E[\hat{f}\,(x)]\right)^2\right] + \sigma_e^2$$

$$Err(x) = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

Proof

Class imbalance

Imbalanced datasets



- With so few examples in data, classification of fraud becomes a needle in a haystack problem.
  - Fraud detection
  - Spam filtering
  - Disease screening
  - SaaS subscription churn
  - …

Confusion matrix - Balanced test set

	Class 1 Predicted	Class 2 Predicted
Class 1 Actual	70	30
Class 2 Actual	10	90

- Classification accuracy: 80%
- Recall (cl.1): 70%
- Precision (cl.1): 87.5%
- F1 (cl.1): 77.8%
- Recall (cl.2): 90%
- Precision (cl.2): 75%
- F1 (cl.2): 81.8%

Confusion matrix - Imbalanced test set

	Class 1 Predicted	Class 2 Predicted
Class 1 Actual	700	300
Class 2 Actual	10	90

- Classification accuracy: 71.8%
- Recall (cl.1): 70%
- Precision (cl.1): 98.6%
- F1 (cl.1): 81.9%
- Recall (cl.2): 90%
- Precision (cl.2): 23.1%
- F1 (cl.2): 36.8%

• Imbalanced Dataset: Classes are not equally represented
• CR goes down, is affected a lot by the majority class
• Precision (and F1) for Class 2 are significantly affected –
- 30% of class 1 examples are misclassified leads to a higher number of FN than TN due to imbalance

Normalized confusion matrix

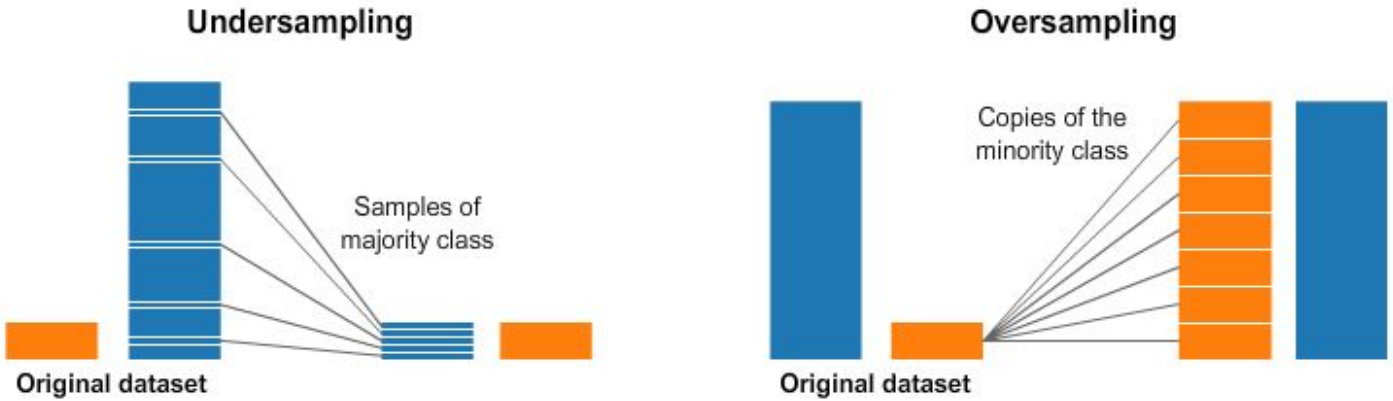
	Class 1 Predicted	Class 2 Predicted
Class 1 Actual	700	300
Class 2 Actual	10	90

Divide by the total number of examples per class →

	Class 1 Predicted	Class 2 Predicted
Class 1 Actual	0.7	0.3
Class 2 Actual	0.1	0.9

- Accuracy can be misleading. It will simply follow the majority class
- F1 is useful as well but is also affected by the class imbalance
- problem.
  - We are not sure if the low score is due to one class being misclassified or class imbalance
- Therefore it's important to examine the confusion matrix

Handling the class imbalance problem

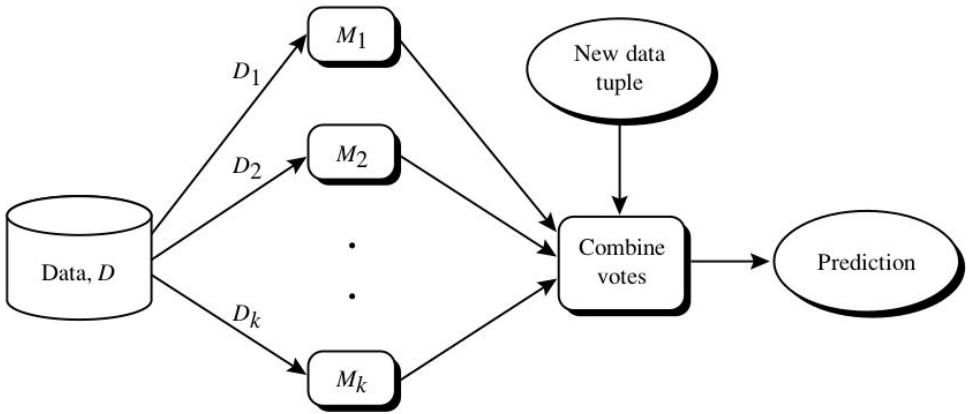


- Upsample the minority class
- Downsample the majority class
  - e.g. select randomly the same number of examples as the minority class.
  - Repeat this procedure several times and train a classifier each time with a different training set.
  - Report the mean and st. dev. of the selected performance measure

Improving ML models with ensembles

Ensemble methods

They combine the decisions from multiple models to improve the overall performance.



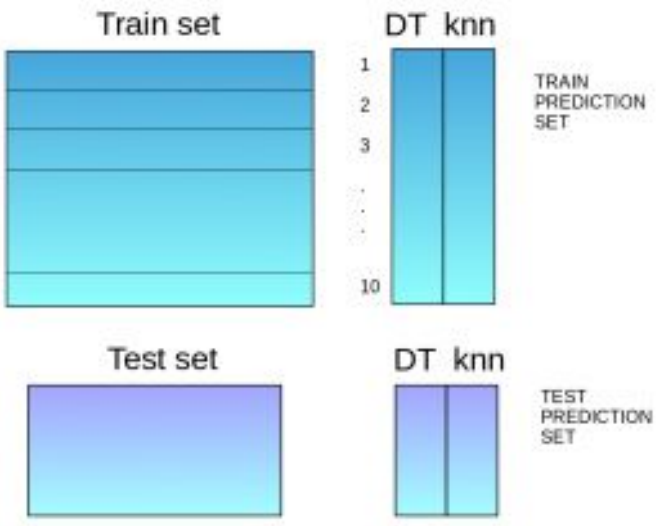
Simple ensembling methods:

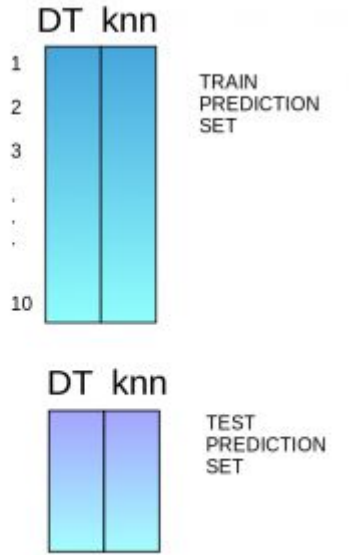
1. Majority voting
2. Averaging (mean or median)
3. Weighted averaging

Advanced ensembling methods:

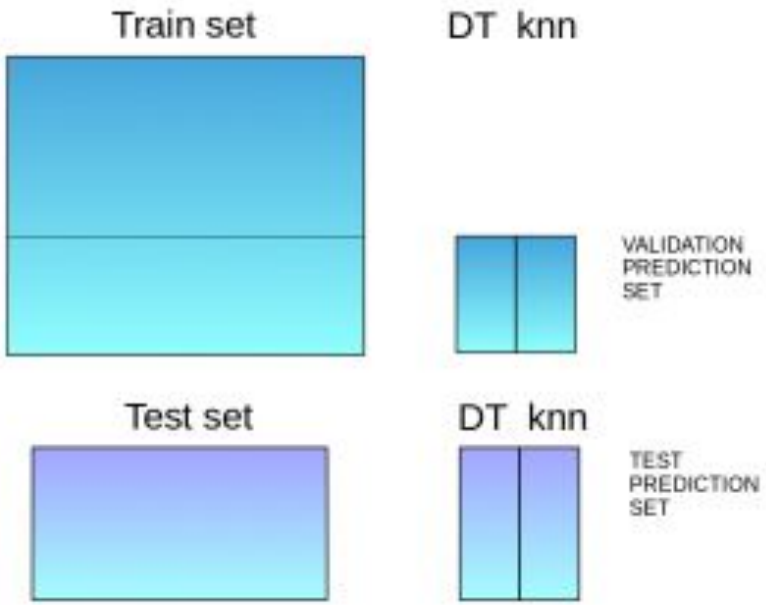
1. Stacking
2. Blending
3. Bagging
4. Boosting

Stacking

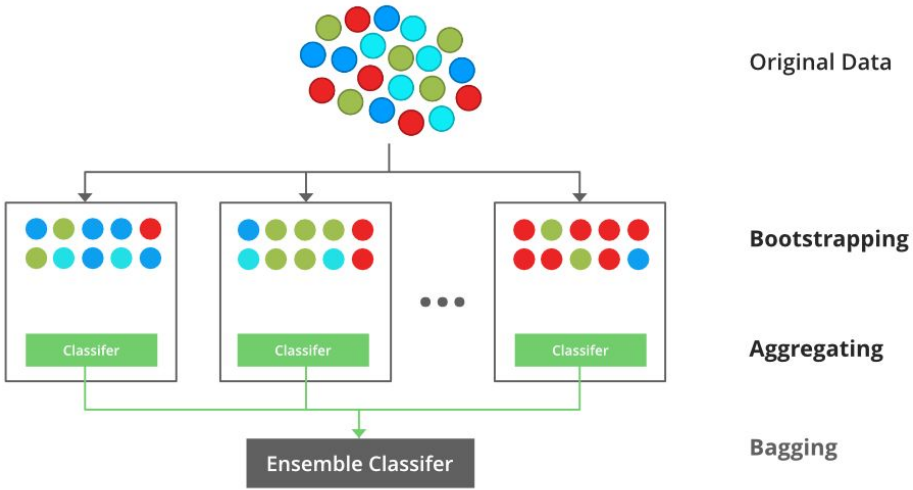

Train set
DT knn
1
2
3
10
TRAIN PREDICTION SET
Test set
DT knn
TEST PREDICTION SET


DT knn
1
2
3
10
TRAIN PREDICTION SET
DT knn
TEST PREDICTION SET

Blending


Train set
DT knn
VALIDATION
PREDICTION
SET
Test set
DT knn
TEST
PREDICTION
SET

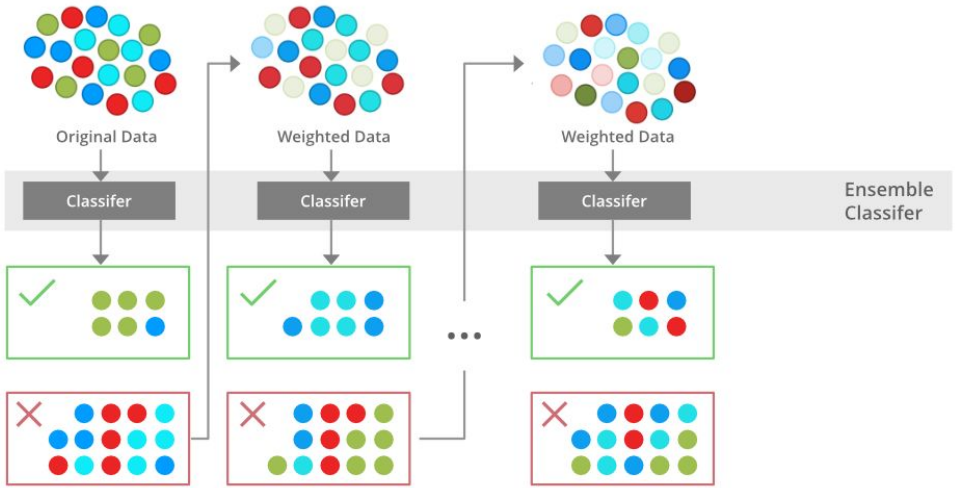
Bagging



- Bootstrap aggregation = bagging (create subsets of observations from the original dataset, with replacement)
- **Majority vote** of multiple classifiers
- Each classifier gets a **bootstrapped sample of the training data**
- Often, significantly better than a single classifier
- Bagging algorithms:
  - Bagging meta-estimator
  - Random forest

Reference (video)

Boosting



- Final result is the weighted average of the multiple classifier results
- The weights depend on the (validation) accuracy of each classifier
- A series of classifiers are iteratively learned
- The weight of each classifier is updated by the current accuracy → iterate again!
- Tends to achieve better accuracy than bagging
- Prone to overfitting
- Boosting algorithms:
  - AdaBoost
  - GBM
  - XGBM
  - Light GBM
  - CatBoost

Reference (video)

Q & A

Research projects are available for interested students.
thanuja@cse.mrt.ac.lk