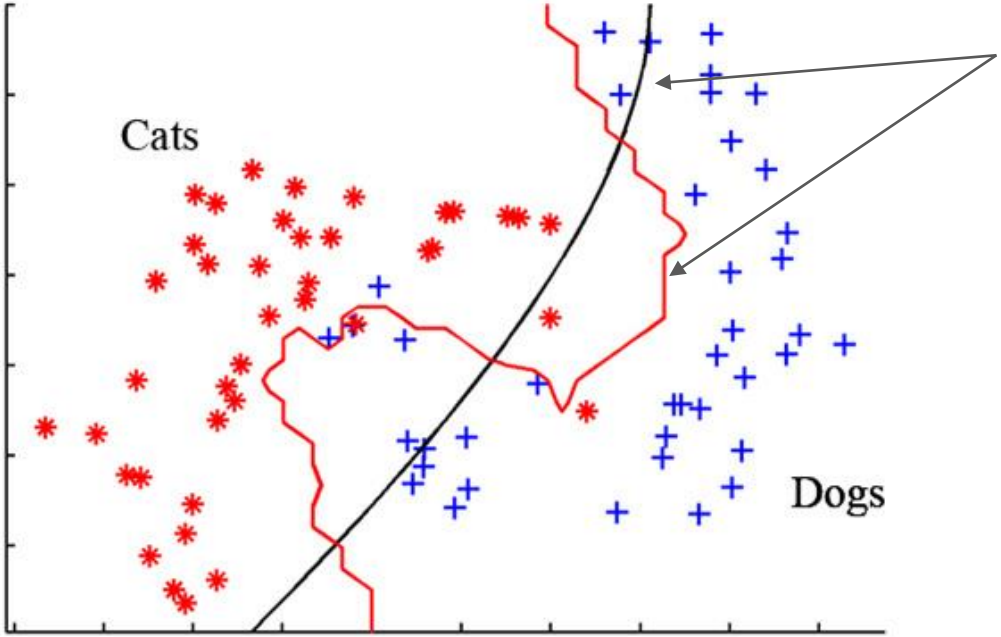
Model Evaluation, and Handling imbalanced data in Machine Learning

Adopted from Bart Baesen's slides on Datacamp (at KU Leuven) and Mehrdad Yazdani's slides (at SF Python talk, Yelp)

Machine learning is just curve fitting

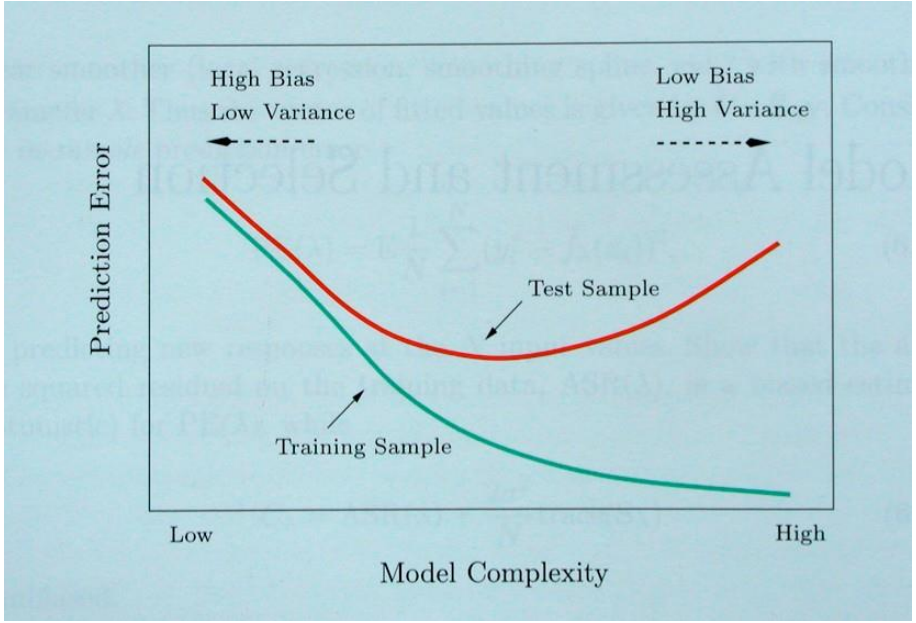


Machine learning algorithm try to “learn” or fit curves that classify data

Note that different algorithms will produce different curves: for example, red curve vs black curve algorithm

Overfitting

- Error on the dataset used to *fit* the model can be misleading
- Doesn't predict future performance.
- Too much complexity can diminish model's accuracy on future data.
- Sometimes called the Bias-Variance Tradeoff.
- A reason why modeling requires subject-matter expertise.



Cross-validation

- In cross-validation the original sample is split into two parts. One part is called the **training** sample, and the other part is called the ***testing*** sample.

- Modeling of the data uses one part only. The model selected for this part is then used to predict the values in the other part of the data. A valid model should show good predictive accuracy.

- Cross validation, by allowing us to have cases in our testing set that are different from the cases in our training set, inherently offers protection against overfitting.

Why Cross Validation?

bigger training set → better **learning**

bigger test set → better **testing**

Key: Train & test sets must be **disjoint.**
And the dataset or sample size is fixed.
They grow at the expense of each other!

⇒ **cross**-validate to maximize both

Cross-validation

What portion of the sample should be in each part?

If sample size is very large, it can be split the sample in 50/50. For smaller samples, split the samples of 2/3, %70/30, %80/20, %90/10

How should the sample be split?

- The most common approach is to divide the sample randomly, thus theoretically eliminating any systematic differences.
- Alternatively, Stratified sampling matches samples proportionally.

Cross Validation – The Ideal Procedure

1.Divide data into two or three sets, training, validation and test sets

2.Find the optimal model on the training set, and use the test set to check its predictive capability

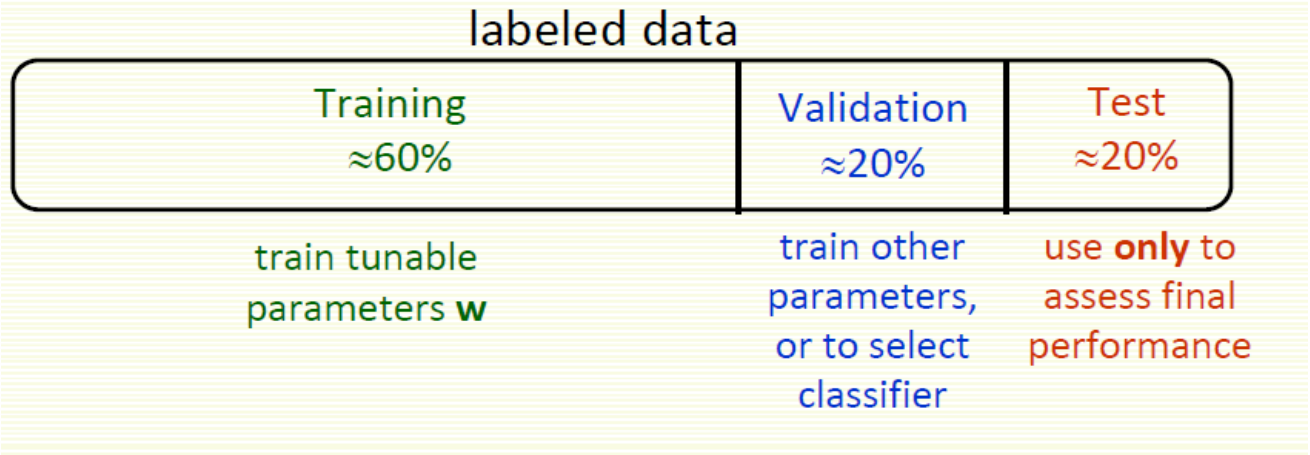
3.See how well the model can predict the test set

4. The validation error gives an unbiased estimate of the predictive power of a model

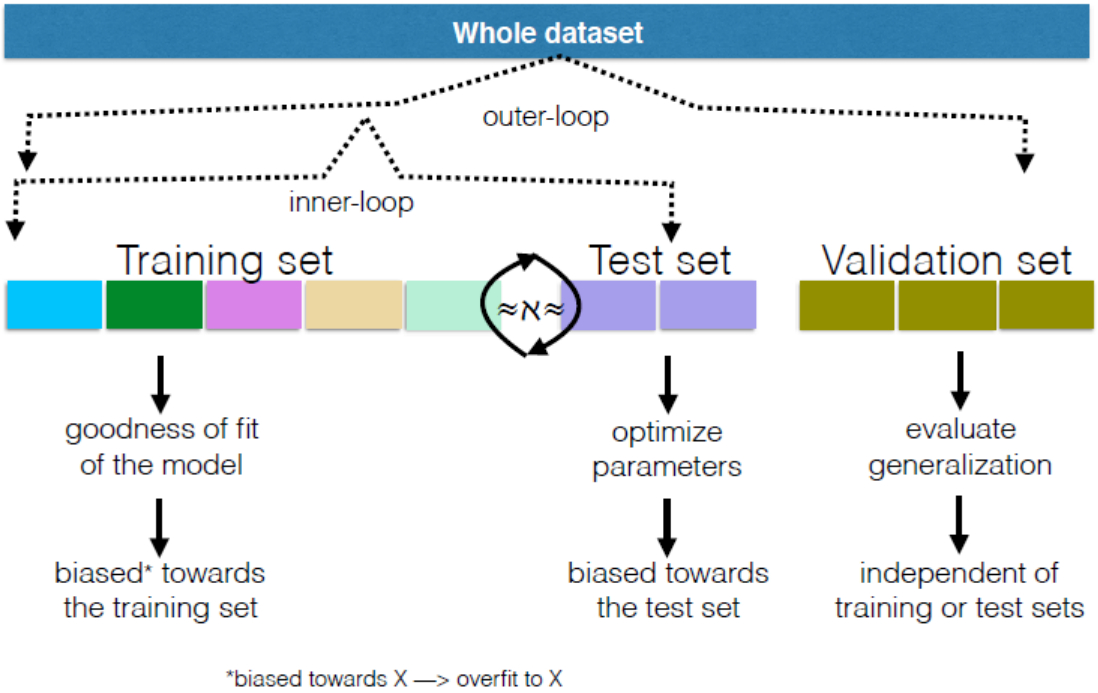
Validation Dataset

Same question when choosing among several classifiers

- our polynomial degree example can be looked at as choosing among 3 classifiers (degree 1, 2, or 3)
- Solution: split the labeled data into three parts



Validation Dataset



Parameters and Hyperparameters

Model Parameters

- These are the entities learned via training from the training data.
- They are not set manually by the designer.
- With respect to deep neural networks, the model parameters are:
  - Weights
  - Biases

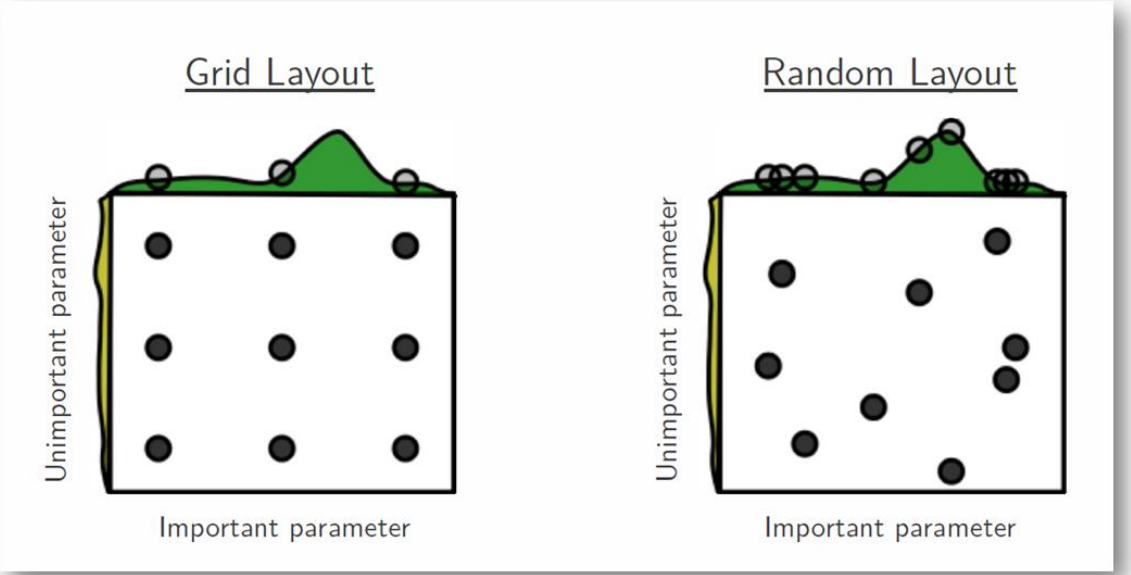
Model Hyperparameters

These are parameters that govern the determination of the model parameters during training

- They are typically set manually via heuristics
- They are tuned during a cross-validation phase

- Examples:
  - Learning rate, number of layers, activation functions, number of units in each layer, many others

Grid Search vs Random Search



Uniform Sampling

Random Sampling

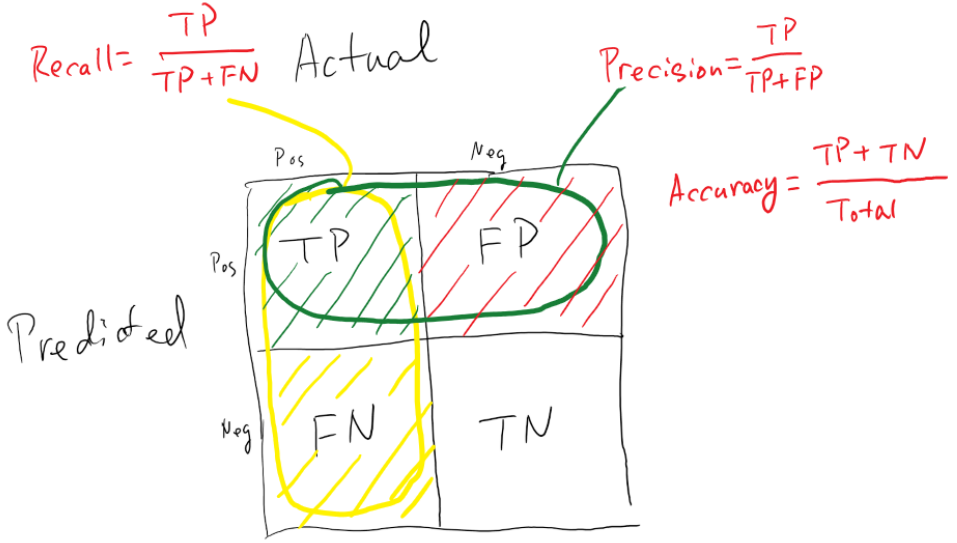
Some hyperparameters won't matter much and others will.
Allows exploring range of hyperparameters more quickly.

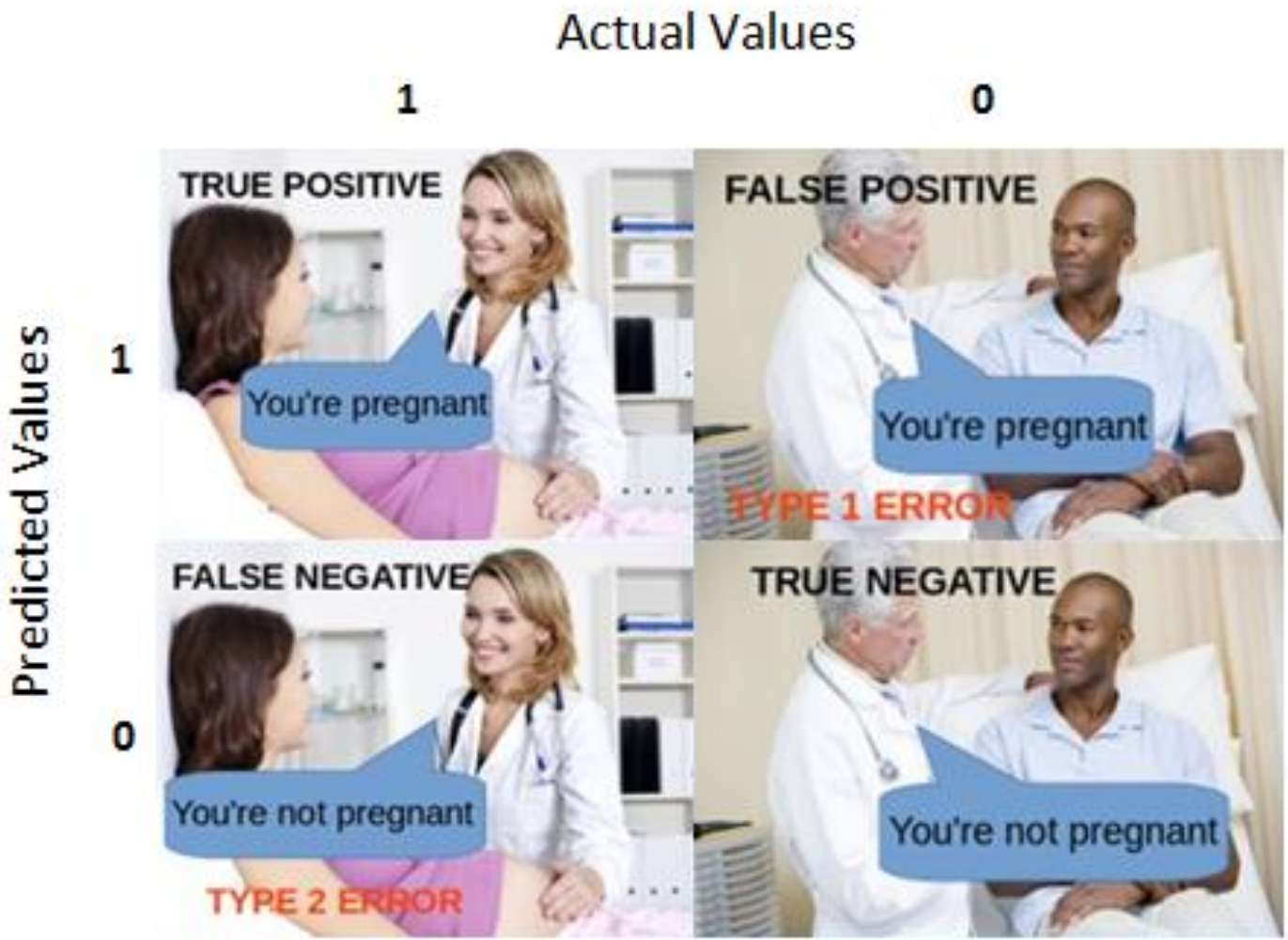
Machine learning model evaluation

Confusion Matrix

```
            Reference
Prediction    0   1
         0  614  14
         1    0   0

Accuracy : 0.9777
```




Actual Values
1
0
Predicted Values
1
0
TRUE POSITIVE
You're pregnant
FALSE POSITIVE
You're pregnant
TYPE 1 ERROR
FALSE NEGATIVE
You're not pregnant
TYPE 2 ERROR
TRUE NEGATIVE
You're not pregnant

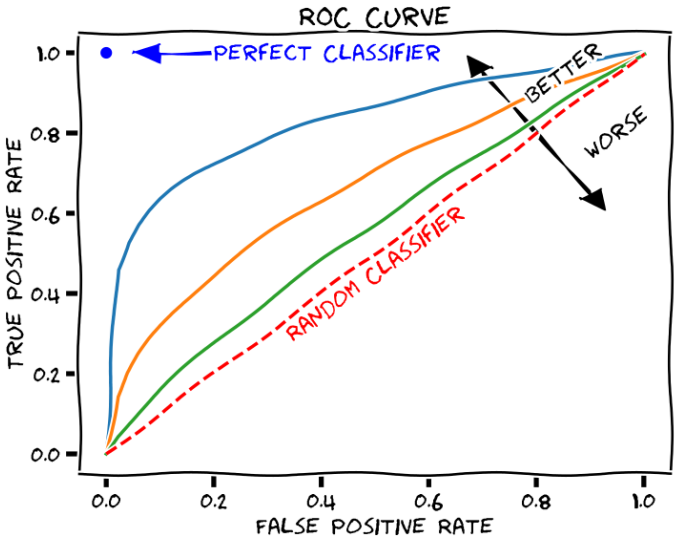
Measuring Error

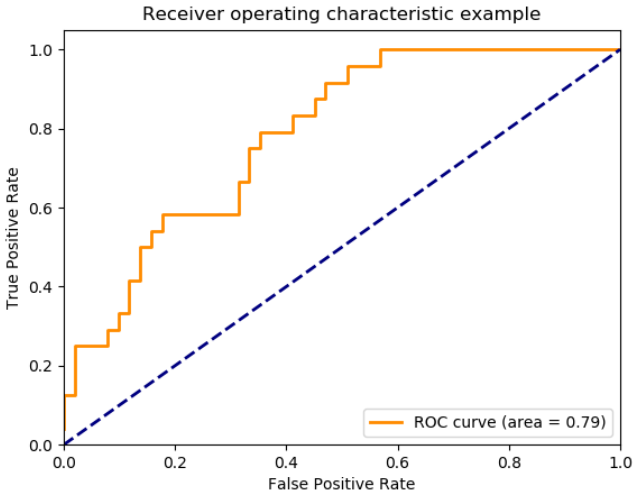
- **Precision** (# of found positives / # of found)
  - # of true positives / (# true positives + # false positives)
- **Recall** (# of found positives / # of positives)
  - Also known as **Sensitivity** (hit rate)
  - TP / (TP+FN)
- **F1 Score**(Harmonic mean of precision and Recall)
  - 2*precision*recall/(precision+recall)

- **Specificity** = TN / (TN+FP) (# of actual negative cases that are correctly identified)
- **Error rate** = # of errors / # of instances = (FN+FP) / N
- **False alarm rate** = FP / (FP+TN) = 1 - Specificity

- A precise classifier is selective
- A classifier with high recall is inclusive

ROC Curve

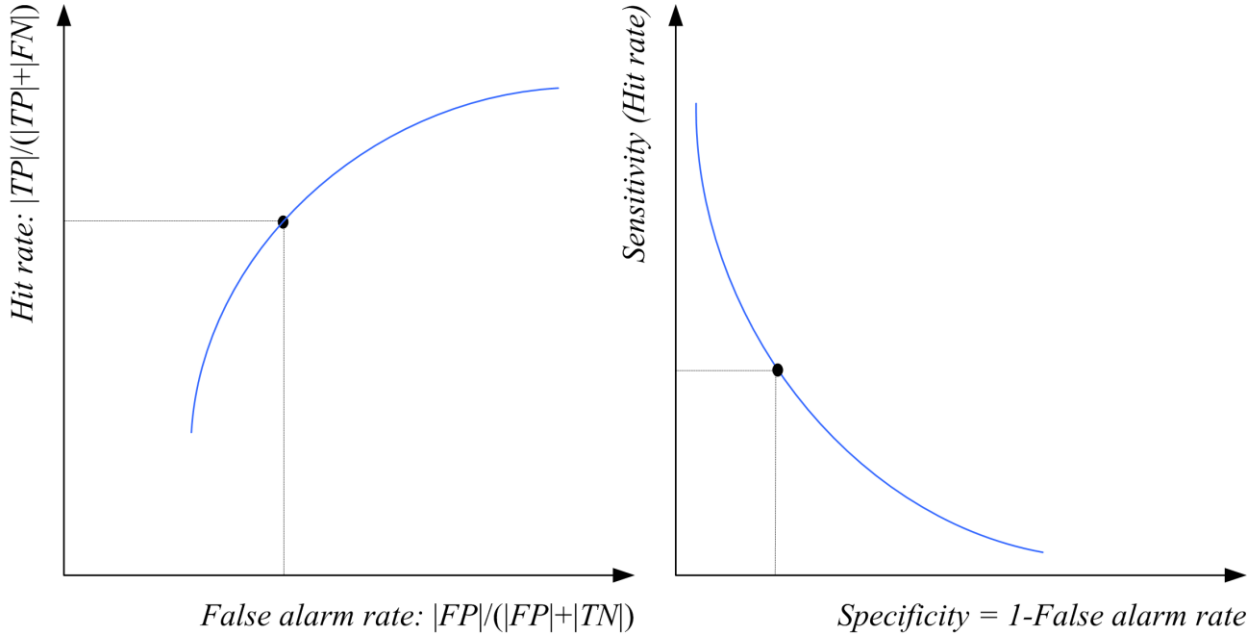




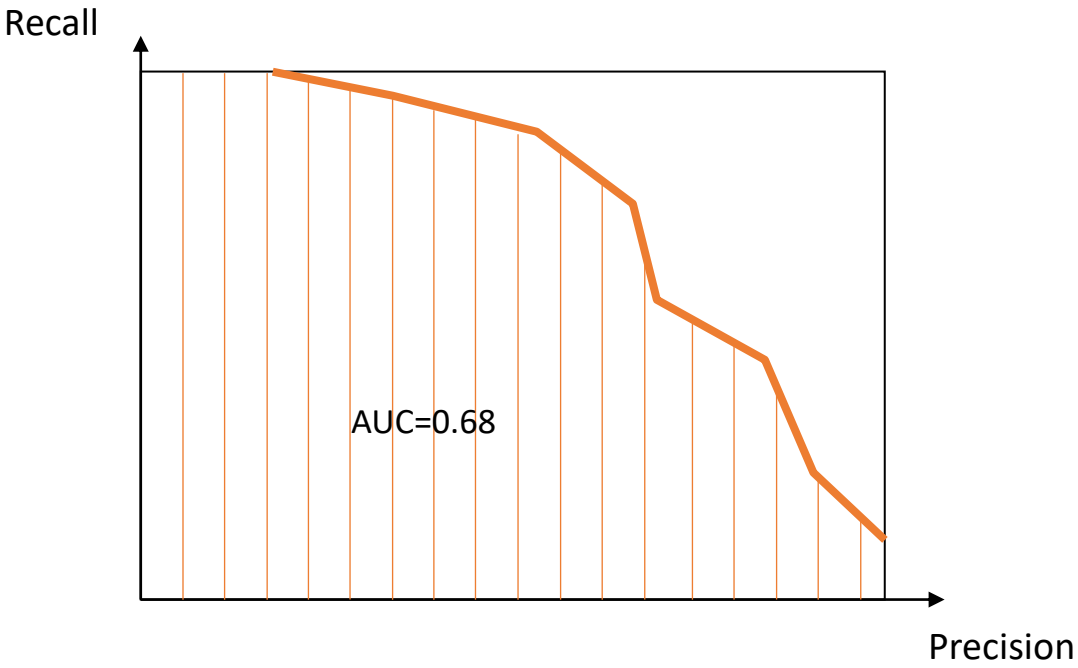
ROC curves are **insensitive** to changes in class distribution, **Class skew independence:**

- If the proportion of positive to negative instances changes in a test set, the ROC curves won't change.
- This is because the metrics TP and FP used for ROC are independent of the class distribution as compared to other metrics like accuracy, precision, etc.

ROC Curve


Hit rate: |TP|/(|TP|+|FN|)
False alarm rate: |FP|/(|FP|+|TN|)
Sensitivity (Hit rate)
Specificity = 1-False alarm rate

Area Under Curve

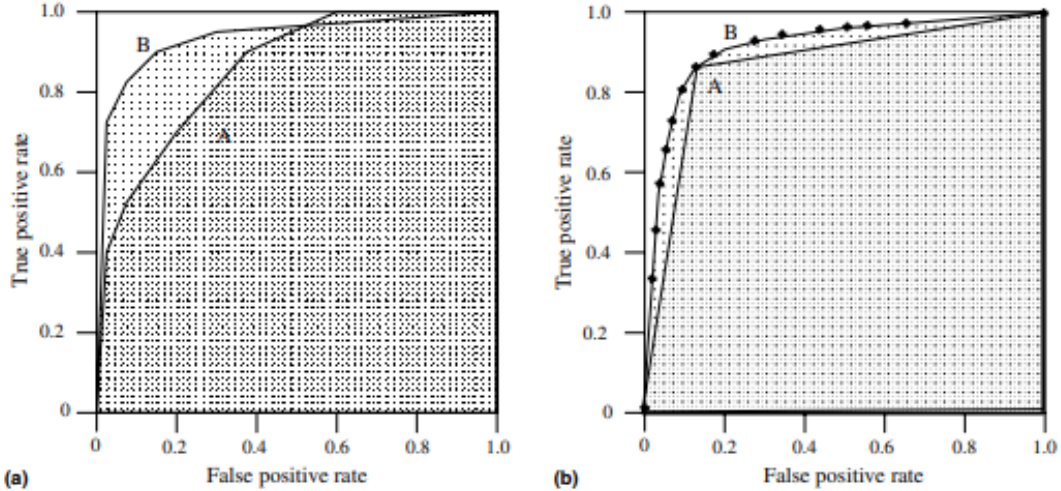


AUC: measure the area under the precision-recall curve

- Since the AUC is a portion of the area of the unit square, its value will always be between 0 and 1.0.
- Random guessing produces the diagonal line between (0, 0) and (1, 1), which has an area of 0.5, no realistic classifier should have an AUC less than 0.5

AUC metrics

- A single number that measures “overall” performance across multiple thresholds
  - Useful for comparing many learners
  - “Smears out” PR curve
- In practice, AUC performs very well and is used when a general measure of prediction is desired.
- Note training / testing set dependence



Accuracy vs Performance

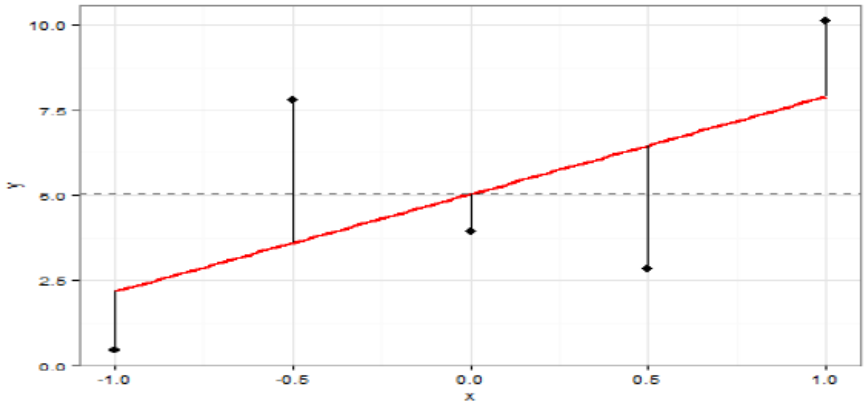
Classifier	Accuracy	Running time
A	90%	80ms
B	92%	95ms
C	95%	1,500ms

REGRESSION MODEL ACCURACY

- How do we determine if one model is predicting better than another model?

The basic relation:

› $Error_i = y_i - f_i$ ← The difference between observed (y) and predicted value (f), when applying the model to unseen data



How to check if a model fit is good?

- The R^2 statistic has become the almost universally standard measure for model fit in linear models.
- What is R^2?

$$R^2 = 1 - \frac{\sum(y_i - f_i)^2}{\sum(y_i - \bar{y})^2}$$

← Model error

← Variance in the dependent variable

- It is the ratio of error in a model over the total variance in the dependent variable.
- Hence the lower the error, the higher the R^2 value.

How to check if a model fit is good?

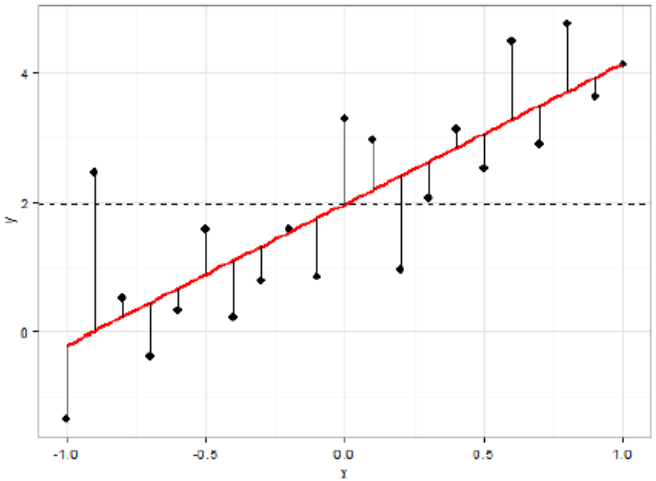
$$\sum(y_i - f_i)^2 = 18.568$$

$$\sum(y_i - \bar{y})^2 = 55.001$$

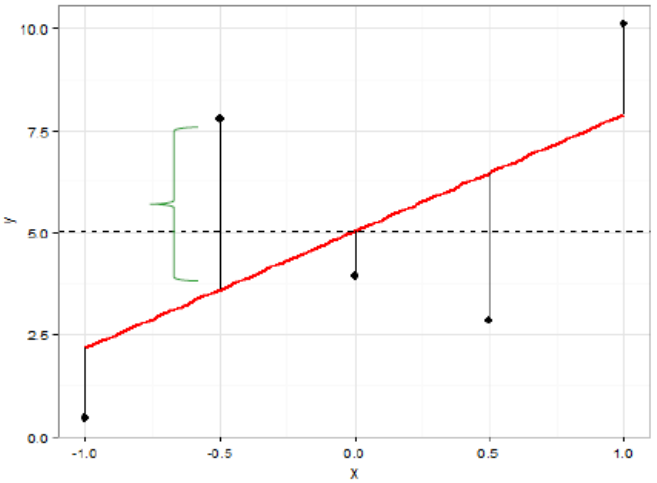
$$R^2 = 1 - \frac{18.568}{55.001}$$

$$R^2 = 0.6624$$

A decent model fit!



- Mean Squared Error (MSE)
  - $1/n\sum(y_i - f_i)^2$
  - 7.96
- Root Mean Squared Error (RMSE)
  - $\sqrt{1/n\sum(y_i - f_i)^2}$
  - 2.82
- Mean Absolute Percentage Error (MAPE)
  - $(1/n\sum|\frac{y_i - f_i}{y_i}|)*100$
  - 120%



MEASURING REGRESSION MODEL ACCURACY

Technique	Abbrev	Measures
Mean Squared Error	MSE	The average of squared errors over the sample period
Mean Error	ME	The average dollar amount or percentage points by which forecasts differ from outcomes
Mean Percentage Error	MPE	The average of percentage errors by which forecasts differ from outcomes
Mean Absolute Error	MAE	The average of absolute dollar amount or percentage points by which a forecast differs from an outcome
Mean Absolute Percentage Error	MAPE	The average of absolute percentage amount by which forecasts differ from outcomes

MEASURING THE MODEL ACCURACY

1. Mean Squared Error

The formula used to calculate the mean squared error is:

$$MSE = \frac{1}{n}\sum_{t=1}^{n}(a_t - f_t)^2$$

2. Mean Percentage Error

The formula used to calculate the mean percentage error is:

$$MPE = \frac{1}{n}\sum_{t=1}^{n}\frac{(a_t - f_t)}{a_t} \times 100$$

3. Mean Absolute Error

The formula used to calculate the mean absolute error is:

$$MAE = \frac{1}{n}\sum_{t=1}^{n}\left|(a_t - f_t)\right|$$

MEASURING THE MODEL ACCURACY

4. Mean Absolute Percentage Error

The formula used to calculate the mean absolute percentage error is:

$$MAPE = \frac{1}{n}\sum_{t=1}^{n}\frac{\left|(a_t - f_t)\right|}{a_t} \times 100$$

Best Practice for Reporting Model Fit

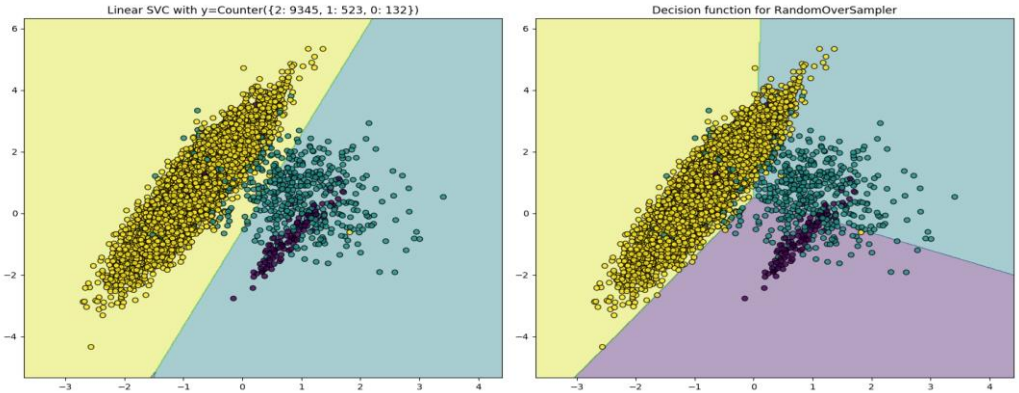
1. Use Cross Validation to find the best model
2. Report the RMSE and MAPE statistics from the cross validation procedure
3. Report the R Squared from the model.

The added cross-validation information will allow one to evaluate not how much variance can be explained by the model, but also the predictive accuracy of the model.

Good models should have a high predictive AND explanatory power!

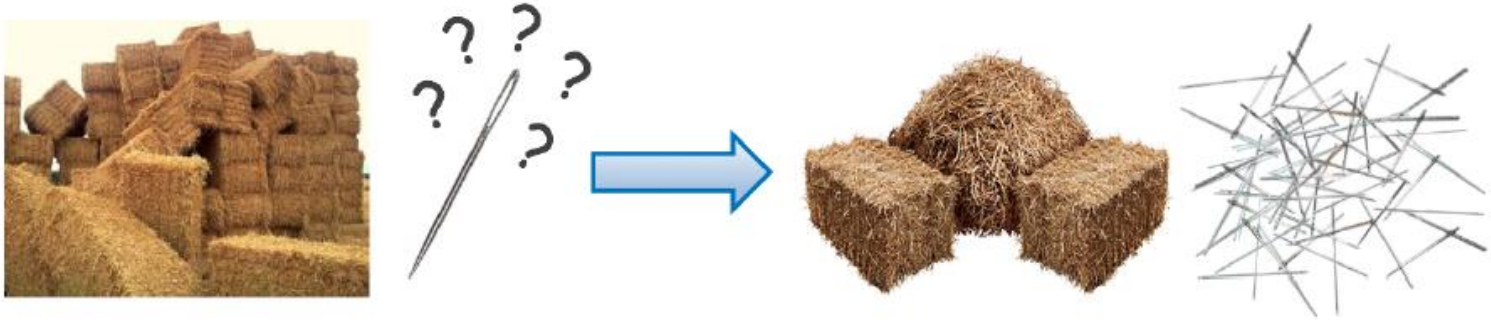
Imbalanced data sets

- **Key challenge** : label events as fraud or not
  - Major challenge for classification methods & anomaly detection techniques
- If you have few examples from a particular class, your decision boundary can miss crucial details!
- Classifier tends to favour majority class (= no-fraud)
  - large classification error over the fraud cases

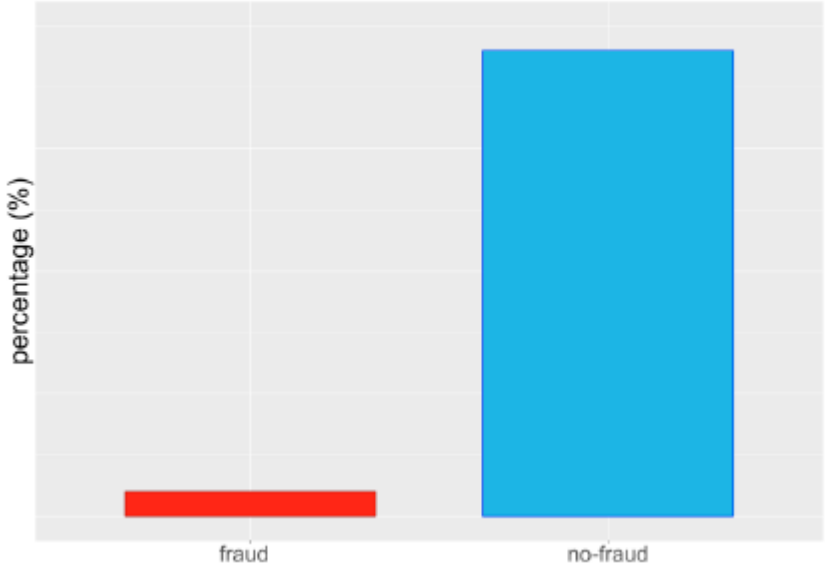


Imbalanced data sets

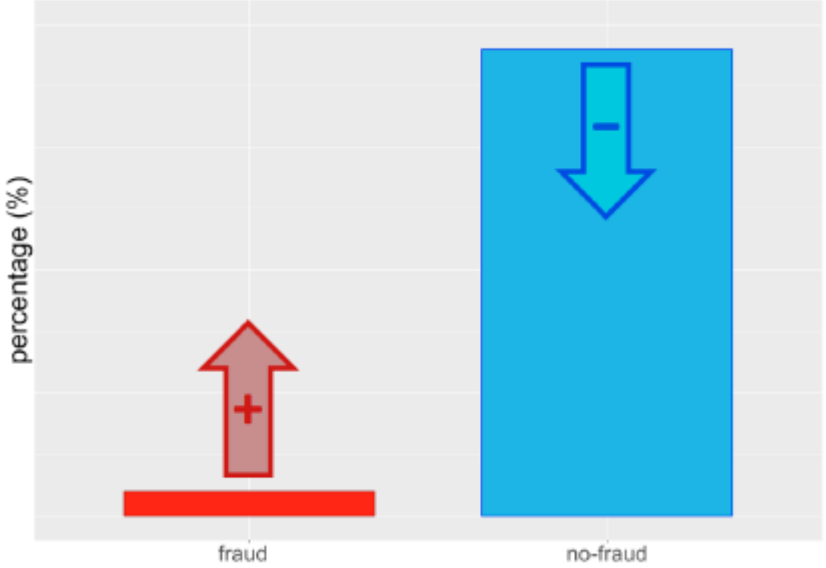
- Classifiers learn better from a balanced distribution
- **Possible solution** : change class distribution with sampling methods
  - Over/under sampling,
  - Synthetic data generation: SMOTE



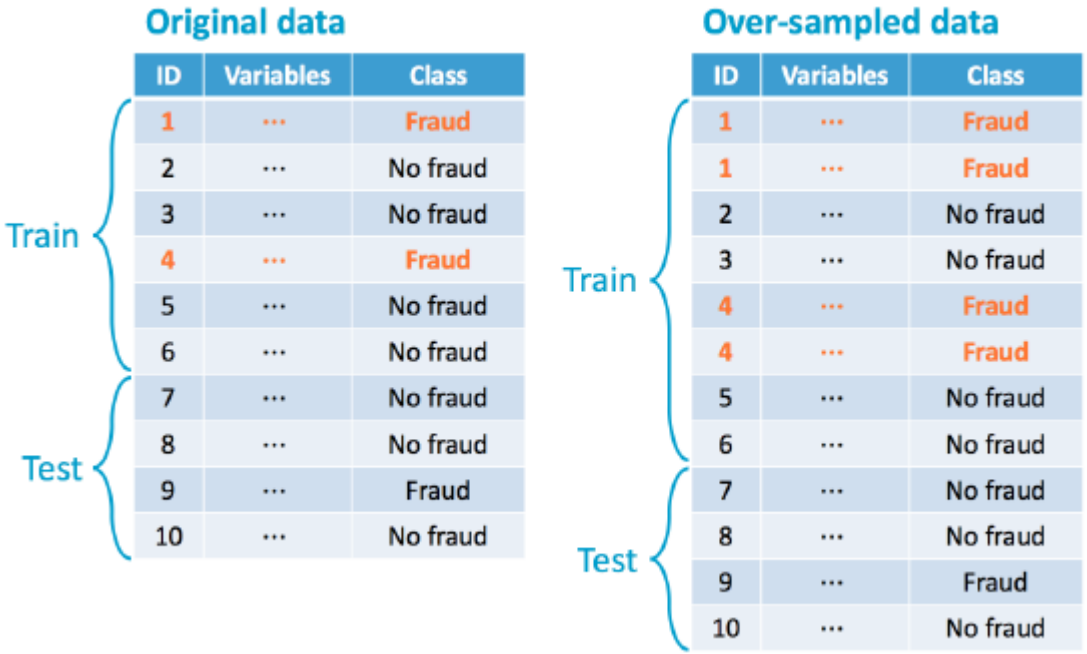
Original data



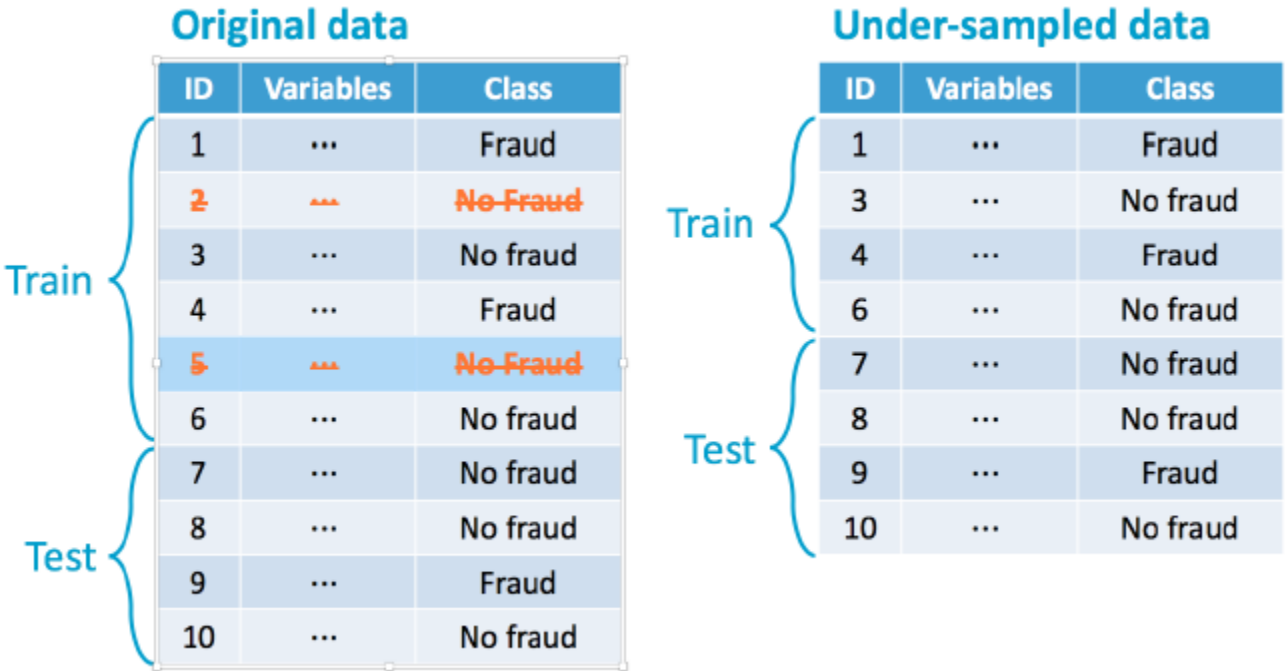
Over-sampling minority class, under-sampling majority class, or both



Random Over-sampling (ROS)



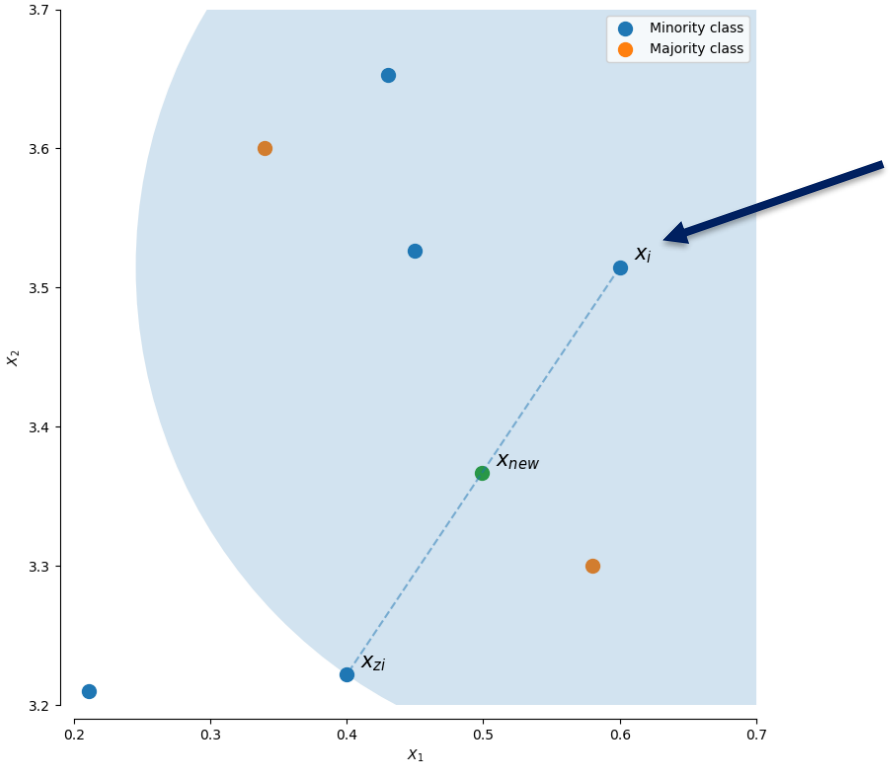
Random under-sampling (RUS)



Imbalanced-learn in Python

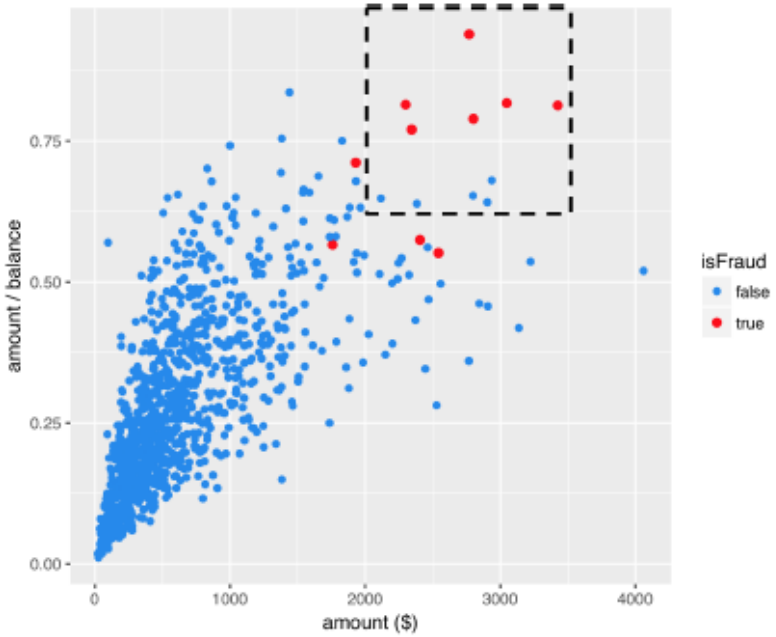
- Contributor library to scikit-learn so follows similar API
- Provides several popular re-sampling techniques
- **Extends scikit-learn pipeline class to prevent data leakage**

SMOTE: Synthetic Minority Over-sampling Technique

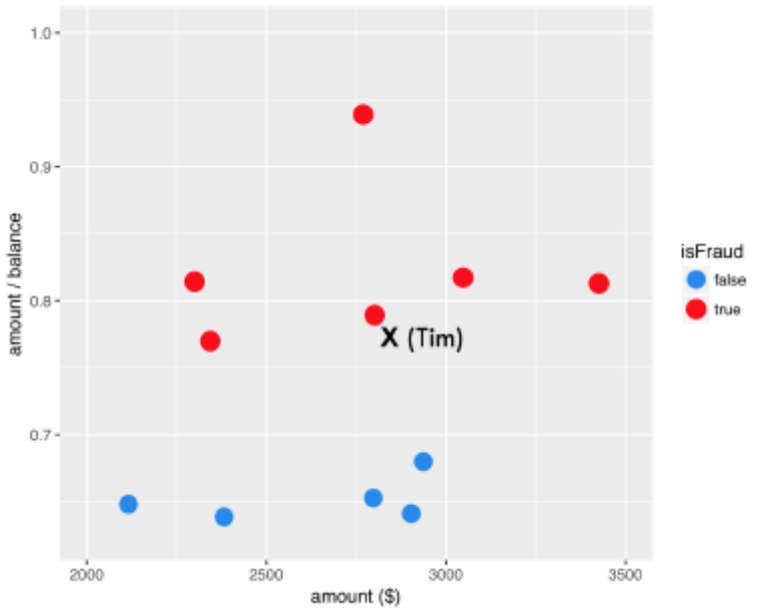


- SMOTE : Synthetic Minority Oversampling TEchnique (Chawla et al., 2002)
- Over-sample minority class (i.e. fraud) by creating synthetic minority cases
- Find “k-nearest neighbors” of an anchor point x_i from minority class
- Randomly select one of the nearest neighbors and interpolate randomly between the two

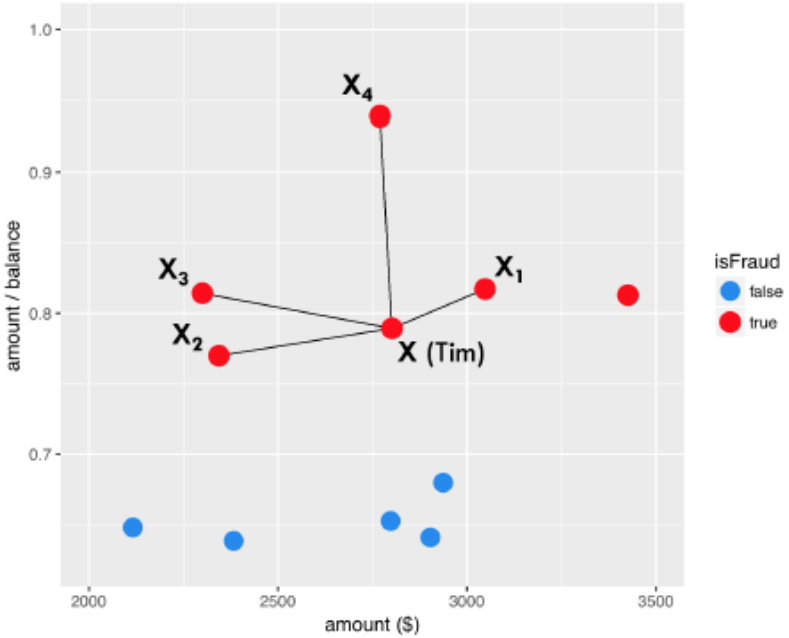
Focus on minority cases



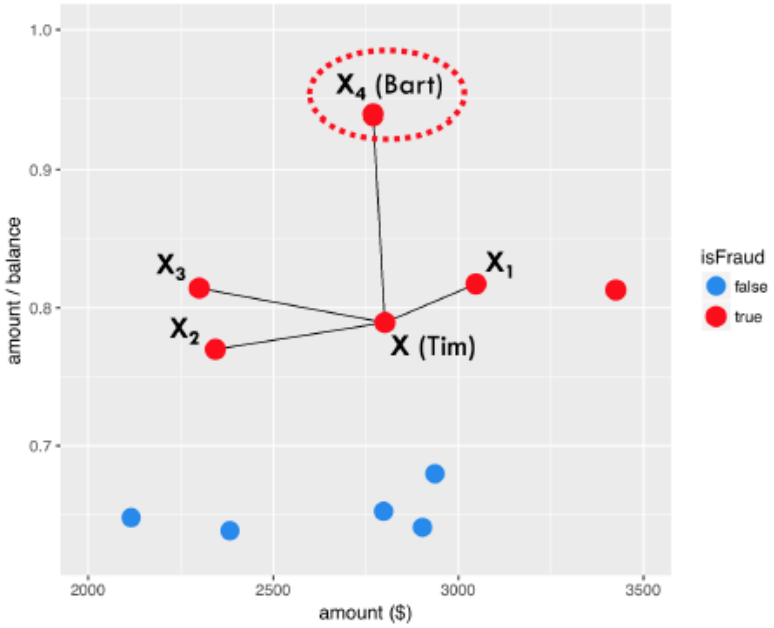
Firstly, select a minority case **X**(Tim)



Step 1
Find K nearest fraudulent neighbors of **X(Tim)**. e.g. K = 4



Step 2
Randomly choose one of Tim's nearest neighbors. e.g. **X4(Bart)**



Step 3
create synthetic sample

X (Tim)

Amount	Ratio
2800	0.79

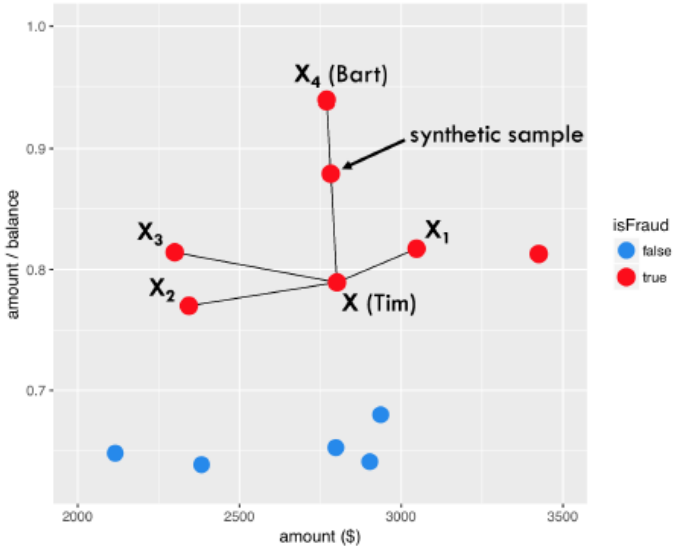
X_4 (Bart)

Amount	Ratio
2770	0.94

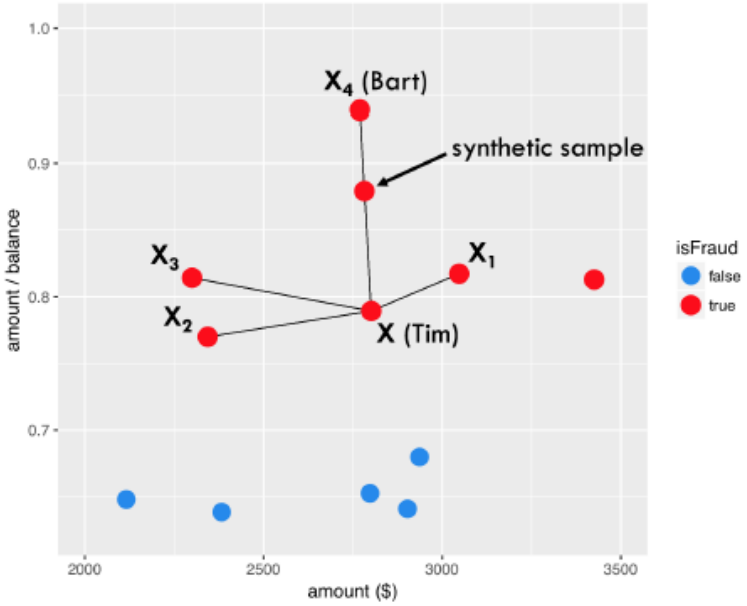
Choose random number between 0 and 1, e.g. **0.6**

Synthetic sample

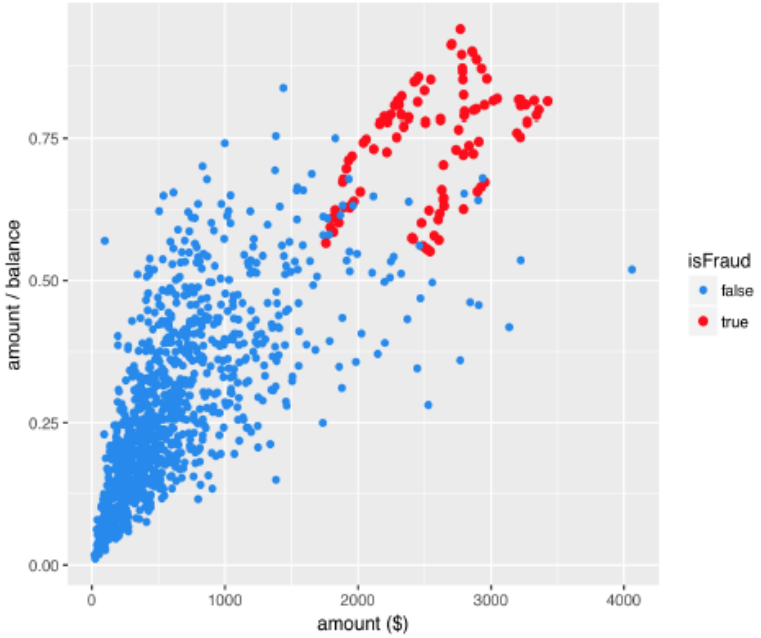
Amount	Ratio
2800 + **0.6** * (2770 - 2800) = **2782**	0.79 + **0.6** * (0.94 – 0.79) = **0.88**

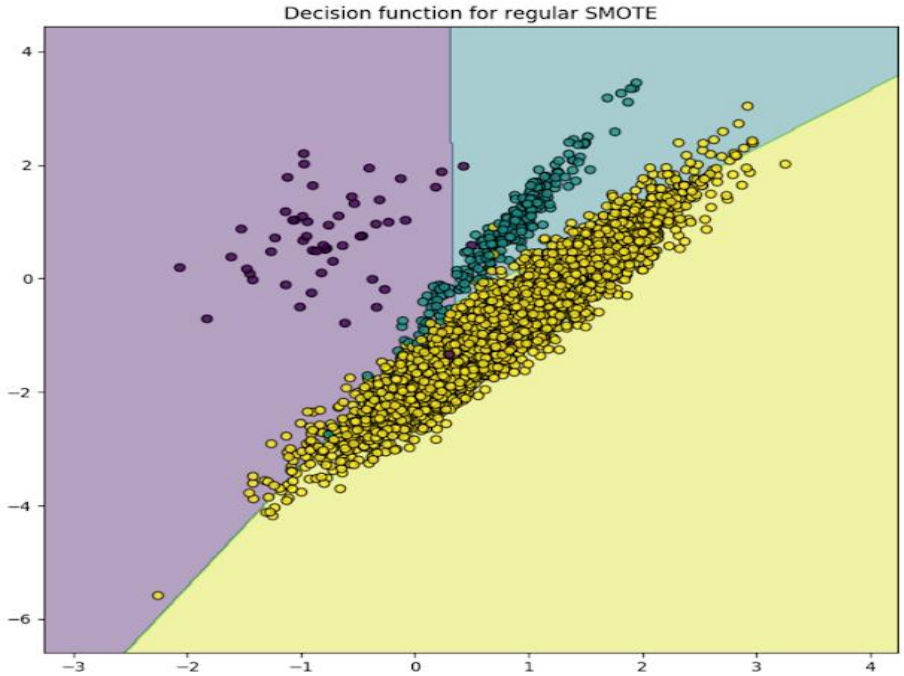


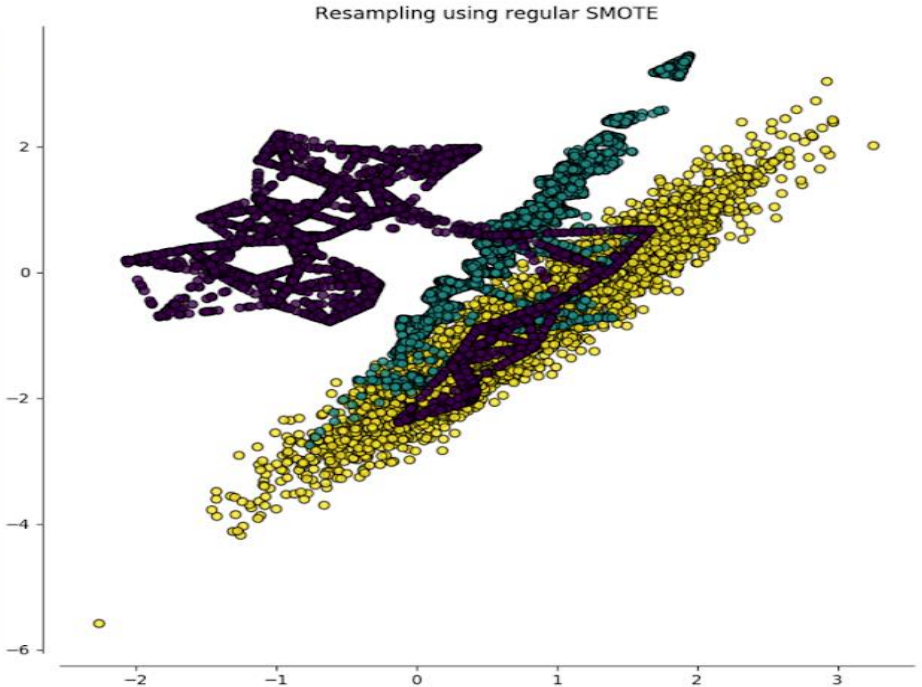
Step 4
Repeat steps 1-3 for each fraud case *dup_size* times. e.g. *dup_size* = 10



Synthetically generated minor cases




Decision function for regular SMOTE
4
2
0
−2
−4
−6
−3
−2
−1
0
1
2
3
4


Resampling using regular SMOTE
2
0
−2
−4
−6
−2
−1
0
1
2
3

Modelling in Python

The model program skeleton would look something like...

```
from sklearn import linear_model

X=[][]          # source data with (n_samples, n_features)

Y=[]            # target value with (n_samples)

clf = linear_model.LinearRegression() # Estimator, or classfier

clf = clf.fit(X, Y) # learn parameters from existing data

Test = [][]             # same shape as X

clf.predict(Test) # predict the target for data in Test
```

1. Instantiate model
2. Fit model with **fit** method
3. Predict with **predict** method

https://www.slideshare.net/TalhaObaid1/machine-learning-101

Inheriting from scikit-learn and writing your own **fit** and **predict** methods are all you need to make your own classifier

Imbalanced-learn in Python

Need to resample with the **fit_sample** to get a balanced dataset

```
sampler = SMOTE()

clf = LinearSVC()

X_resample, y_resampled = sampler.fit_sample(X, y)

clf.fit(X_resample, y_resampled)
```

Imbalanced-learn in Python

```
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import PCA
from sklearn.li import KNeighborsClassifier as KNN
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline

## X_train, y_train


pca = PCA()
smt = SMOTE(random_state=42)
clf = LgisticRegression()
pipeline = Pipeline([('smt', smt), ('pca', pca), ('lr', clf)])

pipeline.fit(X_train, y_train)
```

Imbalanced-learn pipeline should have a classifier with a fit method at the very end