Module 1 :

Machine Learning Review

Supervised Learning Algorithms

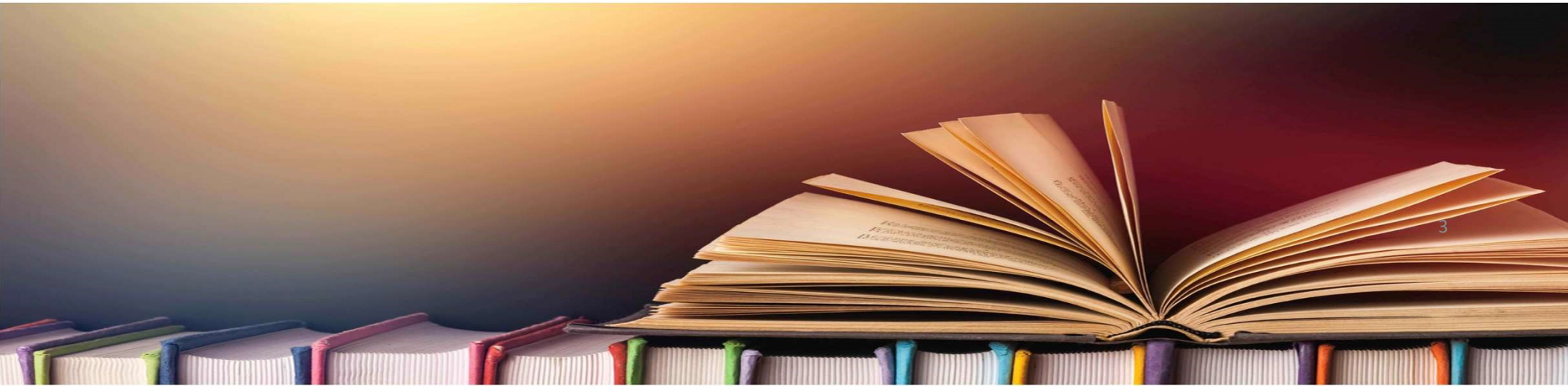
Géraldine Conti, August 2020

Discussion Session

- Review of Notebook 1 (data preparation)
- Based on 02_end_to_end_machine_learning_project.ipynb (A. Geron)
  - Visualize data
  - Correlation matrix
  - Prepare data
  - Encoding
- Exercise (summary of algorithm jungle)

Bibliography

- Deep Learning book (Goodfellow, Bengio, Courville)
- Machine Learning @ Stanford (Prof Andrew Ng)
- Hands-On Machine Learning with Scikit-Learn & Tensorflow (Aurélien Géron)



Learning Objectives

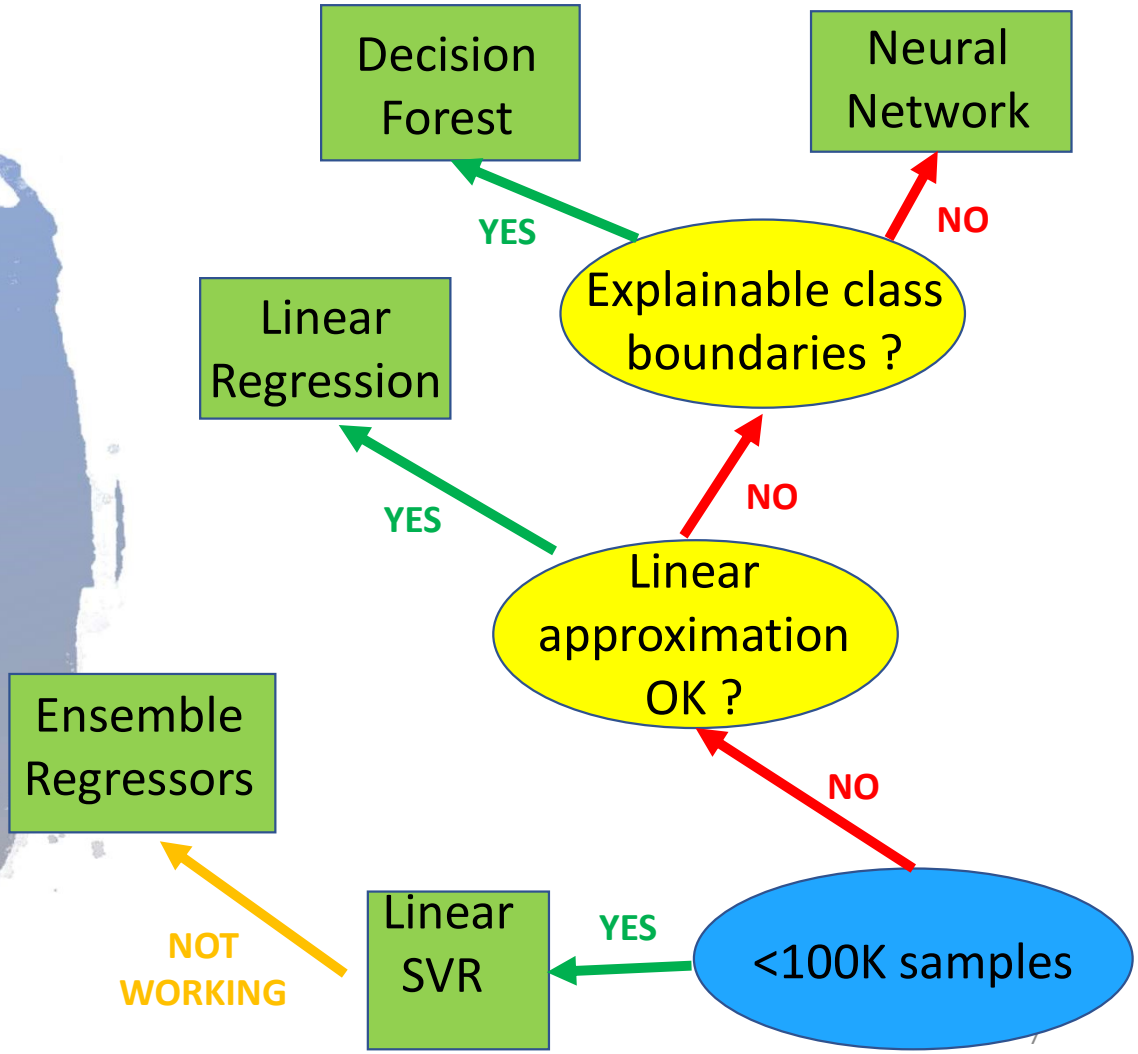
- Regression
  - Linear, polynomial
  - Ridge, LASSO, Elastic Net
  - Performance evaluation
- Classification
  - Logistic regression
  - Naïve Bayes
  - K-nearest neighbors
  - Performance evaluation
- Support Vector Machines (regression/classification)
  - SVC, SVR
- Ensemble methods (regression/classification)
  - Decision trees, random forests
  - Bagging, boosting

Regression

$82000

$55500

???



Measurement Variables

Type	Examples
Nominal	
Ordinal	
Interval	
Ratio	
Dichotomous	

Introduction

- Both a statistical algorithm and a ML algorithm
- Find a linear relationship between :
  - a target (dependent, endogenous)
  - one or more predictors (independent, exogenous)

	Regression	Dependent variable	Independent variable
1	**Simple**	1 (interval, ratio)	1 (interval, ratio, dichotomous)
2	**Multiple**	1 (interval, ratio)	**2+** (interval, ratio, dichotomous)
3	**Logistic**	1 (**dichotomous**)	2+ (interval, ratio, dichotomous)
4	**Ordinal**	1 (**ordinal**)	1+ (nominal, dichotomous)
5	**Multinomial**	1 (**nominal**)	1+ (interval, ratio, dichotomous)
6	**Discriminant**	1 (**nominal**)	1+ (interval, ratio)

- Part of these regression algorithms are actually used for classification

Linear Regression

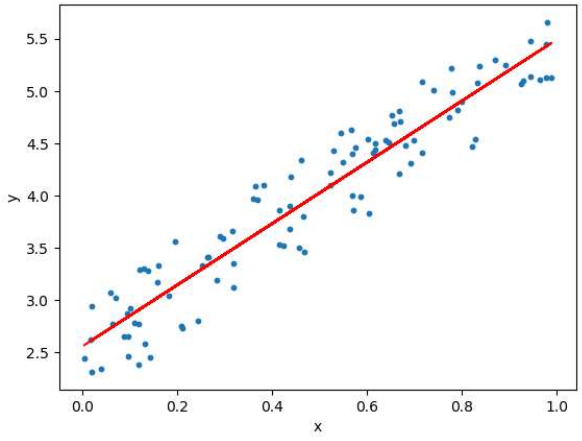
- Regression equation (simplest form) : $y = b \cdot x + c$
  - Complexity = number of coefficients used in the model

- Cost function

$$MSE(X, \hat{y}) = \frac{1}{m}\sum_{i=1}^{m}\left(\theta^T \cdot x^{(i)} - y^{(i)}\right)^2$$

$$\hat{y} = \theta^T \cdot x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

- Can be solved using Gradient descent



Linear Regression in practice

```
>>> import numpy as np
>>> from sklearn.linear_model import LinearRegression
>>> X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
>>> # y = 1 * x_0 + 2 * x_1 + 3
>>> y = np.dot(X, np.array([1, 2])) + 3
>>> reg = LinearRegression().fit(X, y)
>>> reg.score(X, y)
1.0
>>> reg.coef_
array([1., 2.])
>>> reg.intercept_
3.0000...
>>> reg.predict(np.array([[3, 5]]))
array([16.])
```

Polynomial Regression

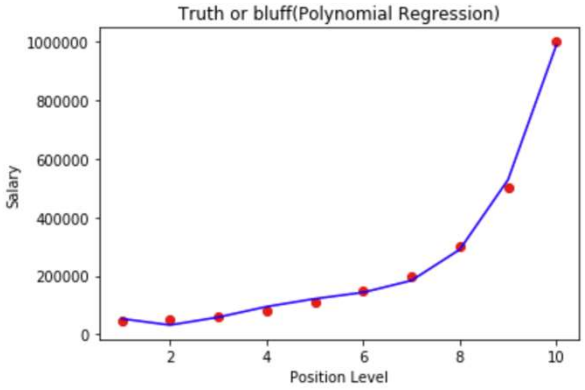
- You can use a linear model to fit nonlinear data

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \cdots + \beta_n x^n + \varepsilon.$$

- Add powers of each feature as new features

- Use LinearRegression on this training data

```
#fitting the polynomial regression model to the dataset
from sklearn.preprocessing import PolynomialFeatures
poly_reg=PolynomialFeatures(degree=4)
X_poly=poly_reg.fit_transform(X)
poly_reg.fit(X_poly,y)
lin_reg2=LinearRegression()
lin_reg2.fit(X_poly,y)
```



Limitations with classical linear regression

- The classical linear regression does not work well with :
  - Multicollinearity : one or more in the independent variables can be expressed as the linear combination of the other independent variables.
  - Number of independent variables > number of observations : the ordinary least square estimates are not valid because there are infinite solutions ot our estimators
- Solution : regularization

Regularized Loss = Loss Function + Constraint

Ridge Regression

- Regularized linear model
- Constraint = half the square of the L2 norm of the weight vector

$$J(\theta) = MSE(\theta) + 0.5 \cdot \alpha \sum_{i=1}^{n} \theta_i^2$$

- Keeps the model weights as small as possible
- α controls how much you want to regularize the model
  - α=0 is a linear regression

LASSO Regression

- Regularized linear model
  - Least Absolute Shrinkage and Selection Operator
- Constraint = L1 norm of the weight vector

$$J(\theta) = MSE(\theta) + \alpha \sum_{i=1}^{n} |\theta_i|$$

- Tends to completely eliminate the weights of the least important features (i.e. set them to zero)

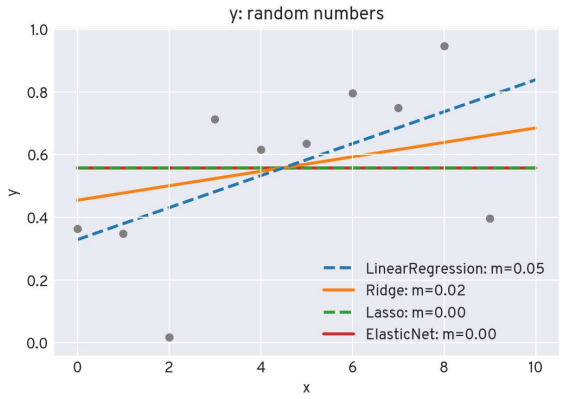
Elastic Net

- Middle ground between Ridge Regression and Lasso Regression
- Constraint = a mix of both Ridge and Lasso's regularization terms

$$J(\theta) = MSE(\theta) + (1-r)/2 \cdot \alpha \sum_{i=1}^{n} \theta_i^2 + r \cdot \alpha \sum_{i=1}^{n} |\theta_i|$$

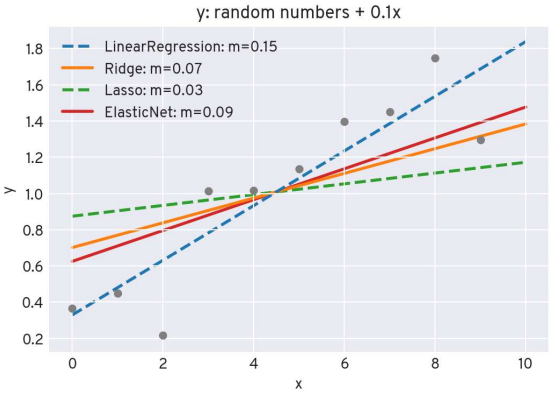
- Use it (or Lasso) if you suspect that only a few features are actually useful

Comparison



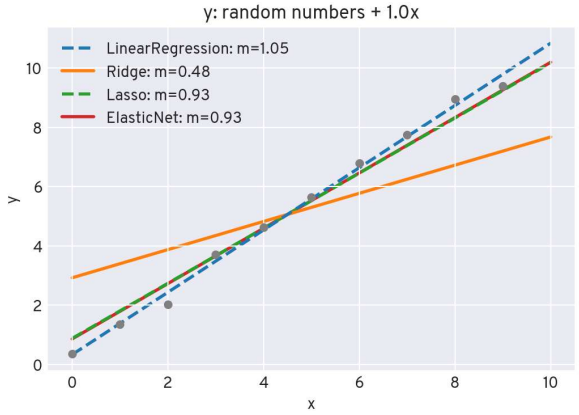
- minor random trend picked up by LR and Ridge
- LASSO, ElasticNet : L1 penalty term high enough to force the weight (slope) to zero when minimizing the loss function.

Small linear component added



- enough for Lasso to not fully "ignore" the slope coefficient anymore.

More linear component added



- Lasso and Elastic Net almost fully "accept" the significant trend
- L2 term (Ridge), leads to a lower slope

https://towardsdatascience.com/regularization-for-machine-learning-models-9173c2e90449

Performance Evaluation

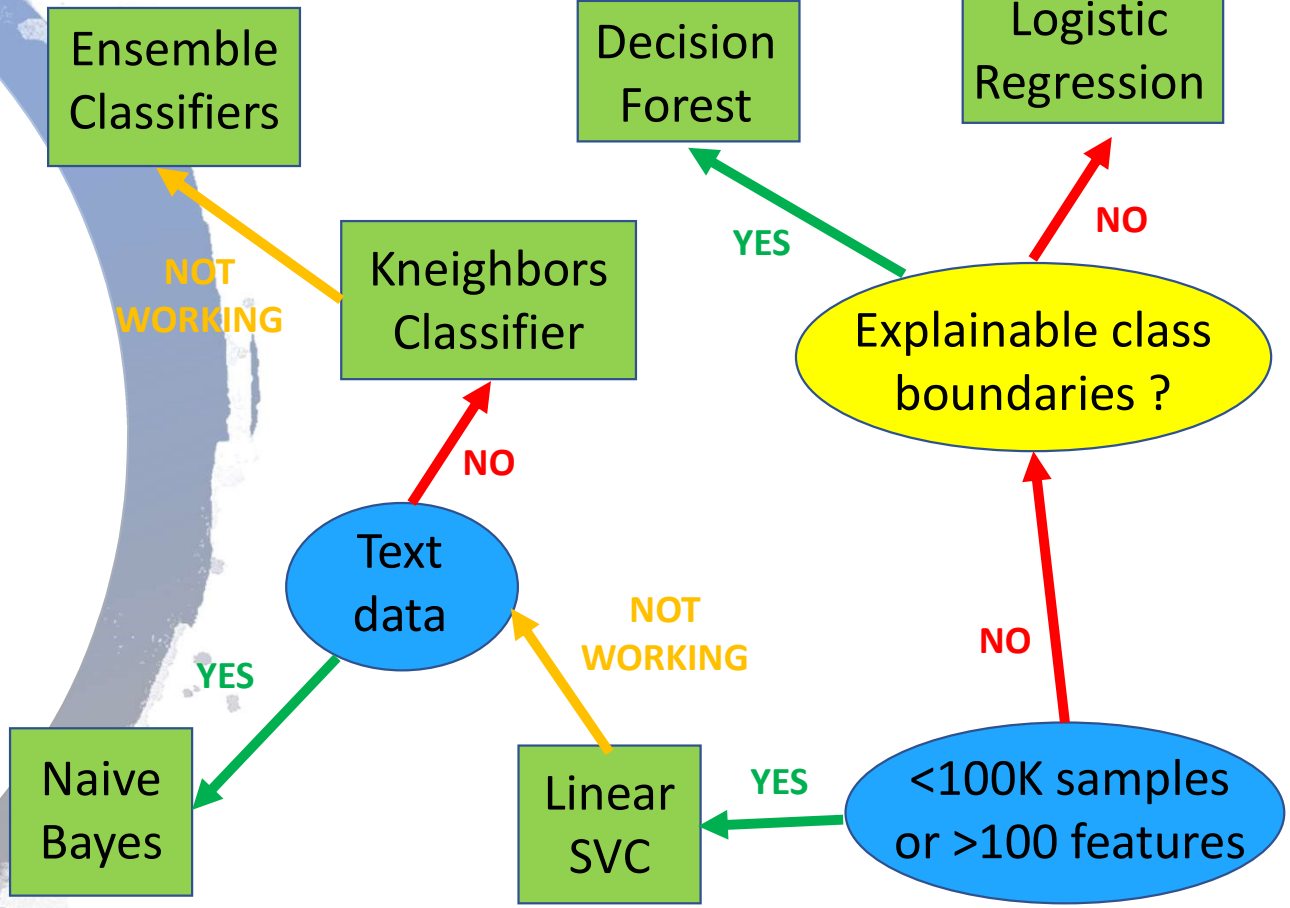
https://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics

Make your own summary table

Classification

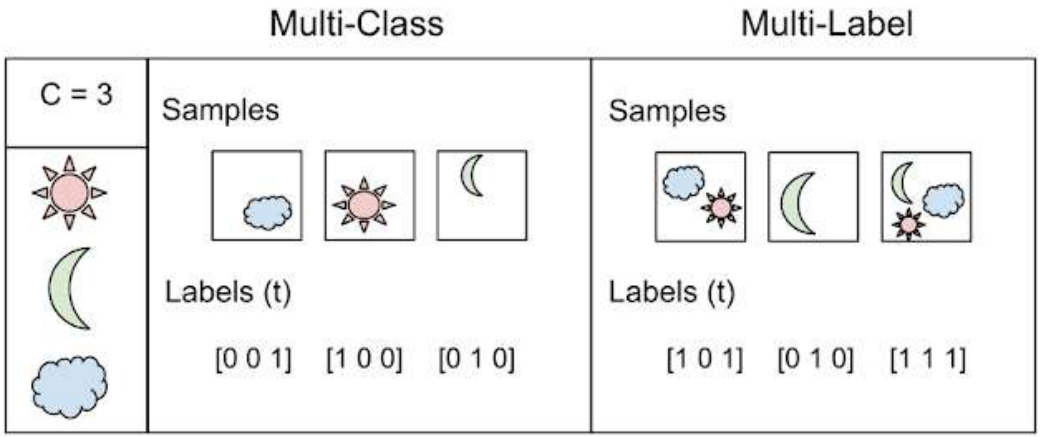
Dog: 96%
Cat: 29%
Duck: 2%
Bird: 0%

Dog: 36%
Cat: 94%
Duck: 2%
Bird: 1%



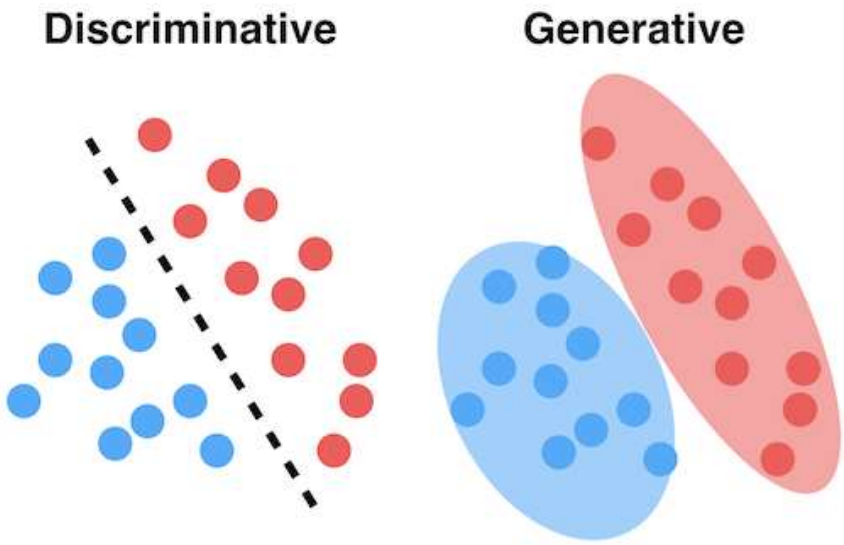
Types of Classifications

- Binary classification
  - Distinction between two classes
- Multiclass classification
  - Distinction between more than two classes
- Multilabel classification
  - Possible to have several classes selected



Types of Models

- Generative :
  - Probabilistic “model” of each class
  - Decision boundary is where one model becomes ore likely
  - Can use unlabeled data
- Discriminative :
  - Focus on the decision boundary
  - Only supervised tasks



1. Logistic Regression
2. K-nearest neighbors
3. Naïve Bayes

1. Logistic Regression

- Mainly used in cases where the output is Boolean
- Data fit into linear regression model, which then be acted upon by a logistic function predicting the categorical target

$$\hat{p} = \sigma(\theta^T \cdot x)$$

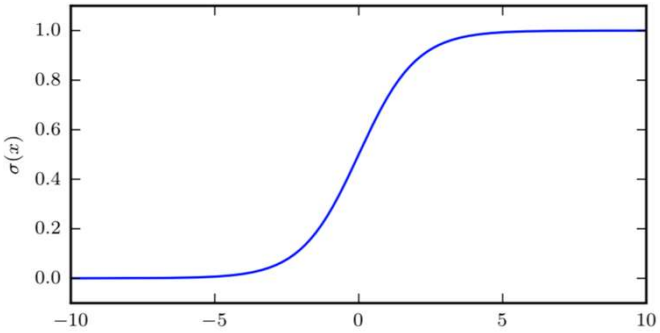
- Decision Boundary : can be linear or non-linear
  - polynomial order increased to get complex decision boundary
- Cost function (convex) : cross-entropy

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(\hat{p}^{(i)}) + (1-y^{(i)})\log(1-\hat{p}^{(i)})]$$

Logistic Function

Sigmoid (*two-class* classifier) :

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



Softmax (*multi-class* classifier) :

$$\text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

Cross-entropy cost function :

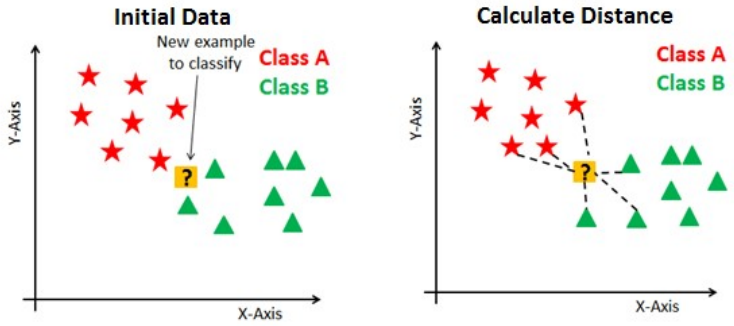
$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} {y_k}^{(i)} \log\left({\hat{p}_k}^{(i)}\right)$$

Logistic Regression in practice

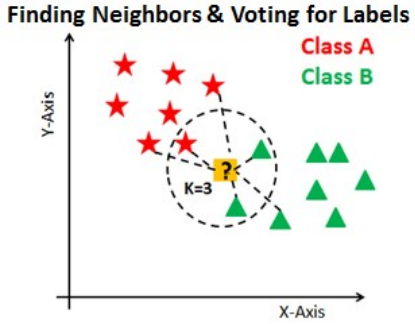
```
>>> from sklearn.datasets import load_iris
>>> from sklearn.linear_model import LogisticRegression
>>> X, y = load_iris(return_X_y=True)
>>> clf = LogisticRegression(random_state=0).fit(X, y)
>>> clf.predict(X[:2, :])
array([0, 0])
>>> clf.predict_proba(X[:2, :])
array([[9.8...e-01, 1.8...e-02, 1.4...e-08],
       [9.7...e-01, 2.8...e-02, ...e-08]])
>>> clf.score(X, y)
0.97...
```

2. K-Nearest-Neighbors (k-NN)

- Algorithm (distance-based) :
  - Take a data point and look at the k closest labeled data points.
  - The data point is assigned the label of the majority of the k closest points



- Steps :
  1) Calculate distance
  2) Find closest neighbors
  3) Vote for labels



K-NN in practice

```
>>> X = [[0], [1], [2], [3]]
>>> y = [0, 0, 1, 1]
>>> from sklearn.neighbors import KNeighborsClassifier
>>> neigh = KNeighborsClassifier(n_neighbors=3)
>>> neigh.fit(X, y)
KNeighborsClassifier(...)
>>> print(neigh.predict([[1.1]]))
[0]
>>> print(neigh.predict_proba([[0.9]]))
[[0.66666667 0.33333333]]
```

Can also be used for multilabel classification

Linear Regression versus K-NN

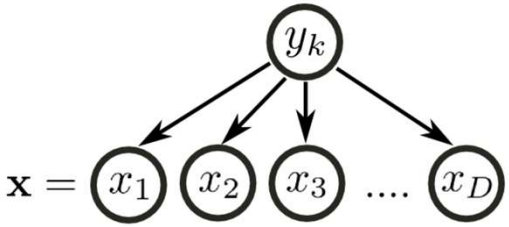
- LR is a parametric approach because it assumes a linear functional form for f(X).
- K-NN is a non-parametric method

	Parametric	**Non-parametric**
Advantages	Easy to fit (small number of coefficients) Easy to interpret	Do not assume an explicit form for f(X)
Disadvantages	Strong assumptions about the form of f(X)	More complex to interpret

- If there is a small number of observations per predictor, then parametric methods then to work better

3. Naive Bayes

- Linear classifier using Bayes theorem and strong independence condition among features
  - "naive" because of the independence of the features
    - Pixels in a digital image
    - Word in a sentence



posterior — likelihood — prior

$$p(y_k|\mathbf{x}) = \frac{p(\mathbf{x}|y_k)p(y_k)}{p(\mathbf{x})}$$

$$p(y_k|\mathbf{x}) = p(y_k)\prod_{i=1}^{D} p(x_i|y_k)$$

choose a suitable distribution depending on the nature of the data, e.g. Gaussian density function

Naive Bayes in practice

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.naive_bayes import GaussianNB
>>> X, y = load_iris(return_X_y=True)
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)
>>> gnb = GaussianNB()
>>> y_pred = gnb.fit(X_train, y_train).predict(X_test)
>>> print("Number of mislabeled points out of a total %d points : %d"
...       % (X_test.shape[0], (y_test != y_pred).sum()))
Number of mislabeled points out of a total 75 points : 4
```

Logistic Regression versus Naive Bayes

- Infinite training size : logistic regression performs better than Naïve Bayes
- Naïve Bayes reaches the asymptotic solution faster (O(logn) than logistic regression (O(n)) : computational cost reduced
- Naive Bayes has a higher bias (because of its assumption on features) but lower variance compared to logistic regression

Performance Evaluation

- To evaluate a ML algorithm, we need a way to measure **how well it performs on the task**
- It is measured **on a separate set** (test set) from what we use to build the function f (training set)
- **Examples** :
  - Classification accuracy (portion of correct answers)
  - Error rate (portion of incorrect answers)
  - Regression accuracy (e.g. least squares errors)

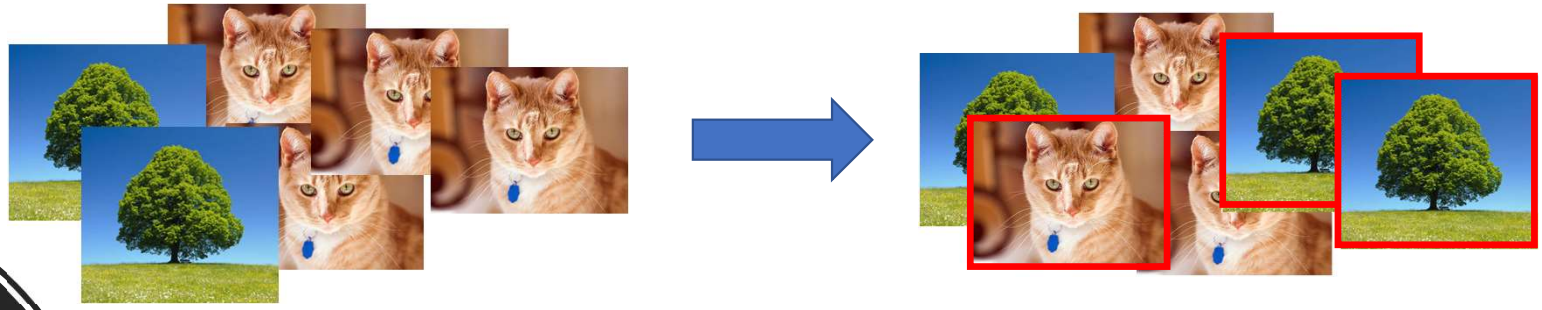
Case Study

- You want to find cats in images
- Classification error (portion of wrong answers) used as an evaluation metric

Algorithm	Classification error (%)
A	**3%**
B	**5%**

Which one is best ?

Evaluation Metrics



- Precision (p)

$$\text{Precision (\%)} = \frac{True\ positive}{Number\ of\ predicted\ positive} x\ 100 = \frac{True\ positive}{(True\ positive + False\ positive)} x\ 100$$

$$\frac{2}{2+1} \times 100 = 66\%$$

- Recall (r)

$$\text{Recall (\%)} = \frac{True\ positive}{Number\ of\ predicted\ actually\ positive} x\ 100 = \frac{True\ positive}{(True\ positive + False\ negative)} x\ 100$$

$$\frac{2}{2+2} \times 100 = 50\%$$

F1-score

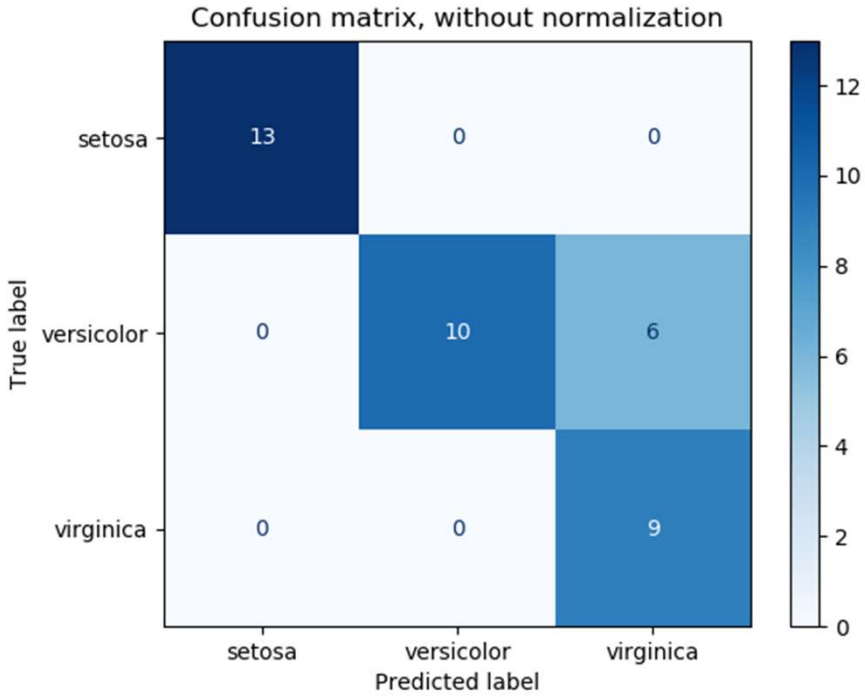
- F1-score is a harmonic mean combining p and r

$$\text{F1-Score} = \frac{2}{\frac{1}{p}+\frac{1}{r}}$$

	Precision	Recall	F1 Score
Algo 1 →	0.5	0.4	0.444 ✓
Algo 2 →	0.7	0.1	0.175
Algo 3 →	0.02	1.0	0.0392

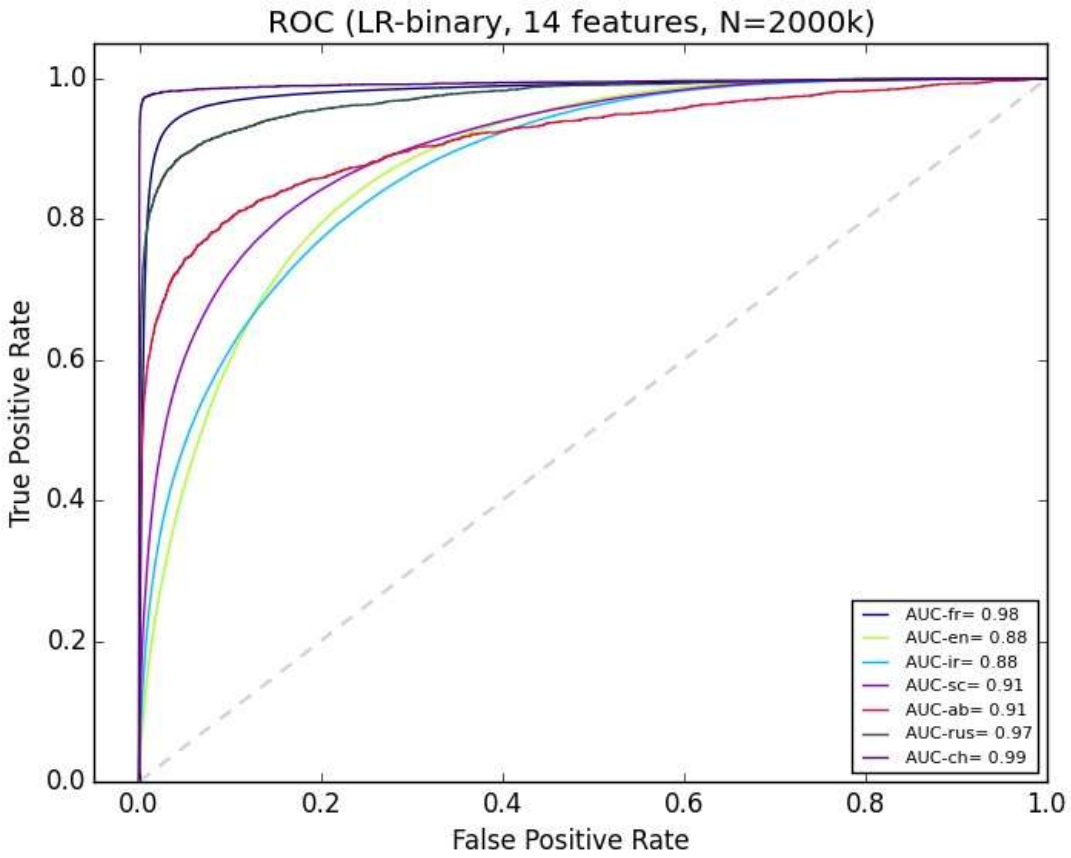
Confusion Matrix

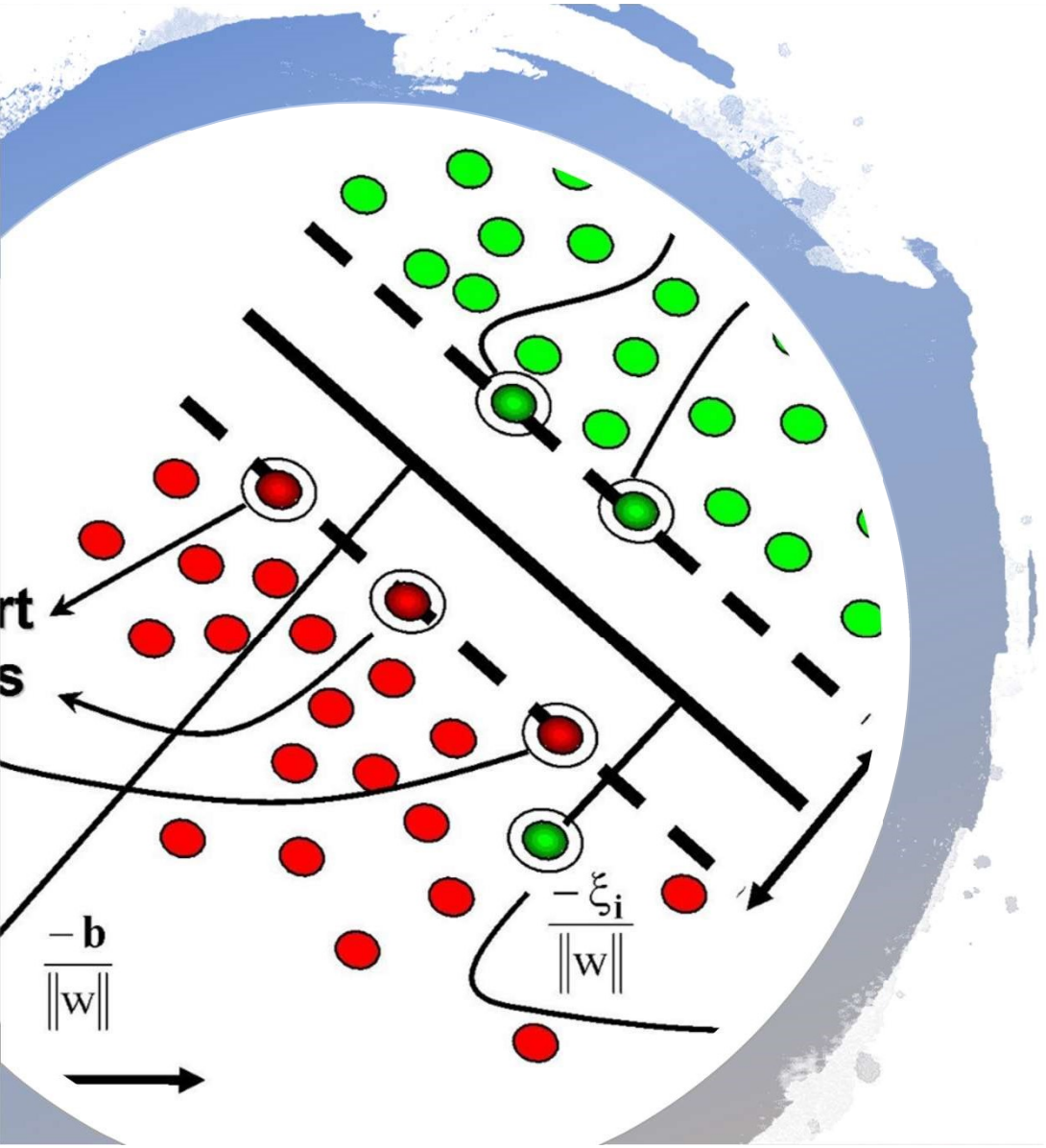
- To evaluate the performance of a classifier
- Count the number of times instances of class A are classified as class B



ROC Curve

- Tool used with binary classifiers for accuracy





Support Vector Machines (SVMs)

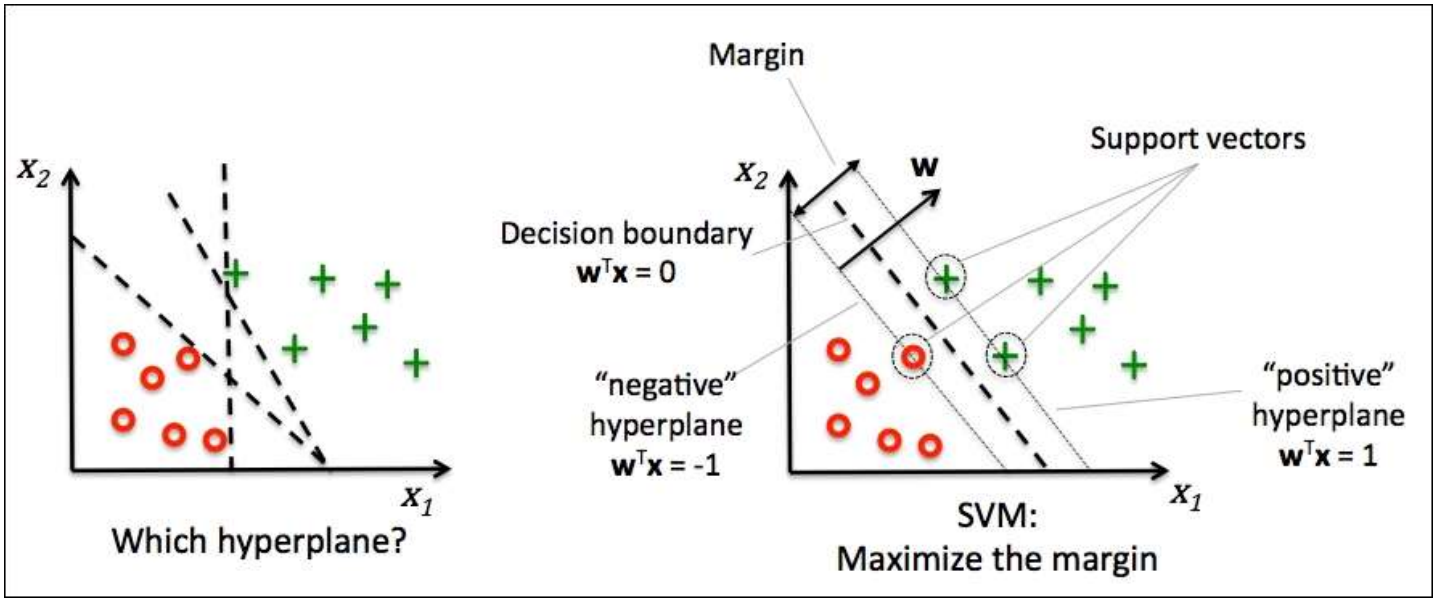
Introduction

- SVM first developed in 1992 for classification (SVC), then generalized to handle regression (SVR)
- Both linear and non-linear cases covered depending on the *choice* of the kernel
- Convex optimization → *unique minimum*
- Suited for complex but small- or medium-sized datasets
- Use cases: object identification, text recognition, bioinformatics, speech recognition,…

Difference with NNs ?

SVC

- "Fit the widest possible street between classes"
  - Large margin classification

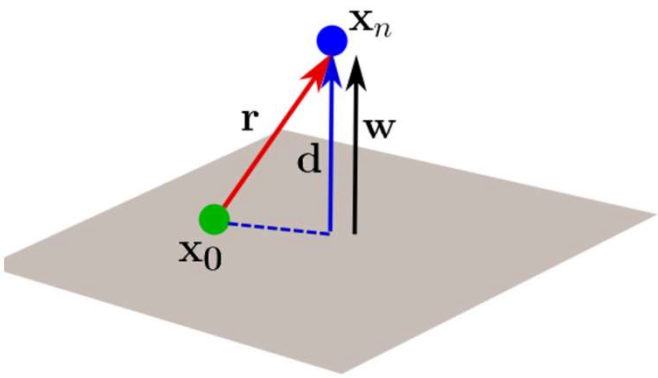


- Strictly binary classifier
- Important to scale the input features

Primal Problem

- Minimization problem:

$$d = \min\left(\frac{1}{2}\|\mathbf{w}\|^2\right) = \min\left(\frac{1}{2}\mathbf{w}^\intercal\mathbf{w}\right)$$



- Constraint :

$$y_n : \text{Given answer} \in \{-1, 1\}$$
$$\hat{y}_n : \text{Predicted answer} \in \{-1, 1\}$$

$$y_n\hat{y}_n \geq 1 \implies y_n(\mathbf{w}^\intercal\mathbf{x}_n + b) \geq 1$$

- Optimization problem :

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^\intercal\mathbf{w} - \sum_{n=1}^{N}\alpha_n\left[y_n(\mathbf{w}^\intercal\mathbf{x}_n + b) - 1\right]$$

- Data points whose $\alpha > 0$ are the support vectors and influence the behavior of the separating hyperplane.

Hard Margin SVM

- Make the equation dependent on one parameter only
- Express part of it as a kernel function **k : dual problem**

$$L(\alpha) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{m=1}^{N} \sum_{n=1}^{N} \alpha_m \alpha_n y_m y_n \mathbf{k}(\mathbf{x}_m, \mathbf{x}_n)$$

$$\text{subject to} \quad 0 \leq \alpha_n \qquad \sum_{n=1}^{N} \alpha_n y_n = 0$$

- Limitation **:**
  - Only works if the data is linearly separable
  - Quite sensitive to outliers

Kernel Examples

- Kernel : function capable of computing the dot product $\varphi(a)^T \cdot \varphi(b)$ based only on the original vectors a and b without having to compute (or even to know about) the transformation φ

- Linear $K(a, b) = a^T \cdot b$

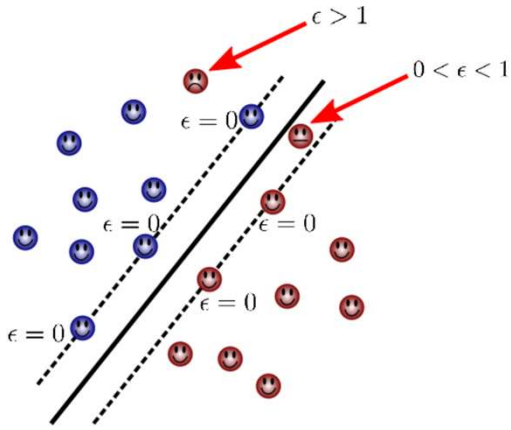
- Polynomial $K(a, b) = \gamma(a^T \cdot b + r)$

- Gaussian RBF $K(a, b) = \exp(-\gamma \|a - b\|^2)$

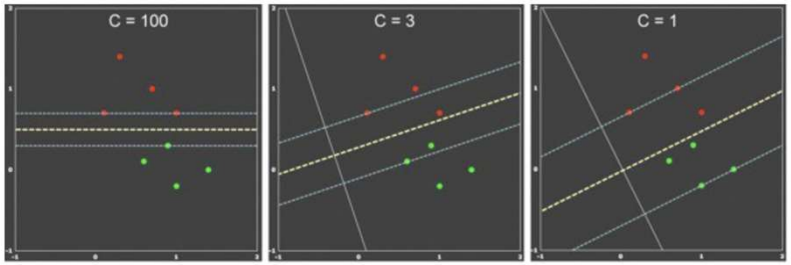
- Disadvantages :
  - Cost of training high with large datasets
  - Generic kernels struggle to generalize well

Soft Margin SVM

- Define a tolerance variable ε to violate the margins to be minimized



- New variable C
  - Large C → small margin
  - Small C → large margin



$$L(\alpha) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{m=1}^{N} \sum_{n=1}^{N} \alpha_m \alpha_n y_m y_n \mathbf{k}(\mathbf{x}_m, \mathbf{x}_n)$$

$$\text{subject to} \quad 0 \leq \alpha_n \leq C, \qquad \sum_{n=1}^{N} \alpha_n y_n = 0$$

SVC in practice

- sklearn.svm.SVC : fit time scales at least quadratically with the number of samples and may be impractical beyond 10000 samples

```
>>> import numpy as np
>>> X = np.array([[-1, -1], [-2, -1], [1, 1], [2, 1]])
>>> y = np.array([1, 1, 2, 2])
>>> from sklearn.svm import SVC
>>> clf = SVC(gamma='auto')
>>> clf.fit(X, y)
SVC(gamma='auto')
>>> print(clf.predict([[-0.8, -1]]))
[1]
```

Kernel function = 'linear', 'poly', 'rbf', 'sigmoid', … only with the SVC class, not the LinearSVC !

C=1 (default value)

- sklearn.svm.LinearSVC : can be used with larger datasets (up to 100000 samples)

```
>>> from sklearn.svm import LinearSVC
>>> from sklearn.datasets import make_classification
>>> X, y = make_classification(n_features=4, random_state=0)
>>> clf = LinearSVC(random_state=0, tol=1e-5)
>>> clf.fit(X, y)
LinearSVC(random_state=0, tol=1e-05)
>>> print(clf.coef_)
[[0.085... 0.394... 0.498... 0.375...]]
>>> print(clf.intercept_)
[0.284...]
>>> print(clf.predict([[0, 0, 0, 0]]))
[1]
```

LinearSVC much faster than SVC(kernel='linear'), as based on the liblinear library

Non-linear SVC

- Add polynomial features
  - Example : 2nd-degree polynomial mapping

Transformed vector is 3dim instead of 2dim !

$$\phi(\mathbf{x}) = \phi\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right) = \begin{pmatrix} x_1^2 \\ \sqrt{2}\,x_1 x_2 \\ x_2^2 \end{pmatrix}$$

- Kernel trick : apply the same mapping, then compute the dot product of the transformed vectors

$$\phi(\mathbf{a})^T \cdot \phi(\mathbf{b}) = \begin{pmatrix} a_1^2 \\ \sqrt{2}\,a_1 a_2 \\ a_2^2 \end{pmatrix}^T \cdot \begin{pmatrix} b_1^2 \\ \sqrt{2}\,b_1 b_2 \\ b_2^2 \end{pmatrix} = a_1^2 b_1^2 + 2a_1 b_1 a_2 b_2 + a_2^2 b_2^2$$

$$= (a_1 b_1 + a_2 b_2)^2 = \left(\begin{pmatrix} a_1 \\ a_2 \end{pmatrix}^T \cdot \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}\right)^2 = \left(\mathbf{a}^T \cdot \mathbf{b}\right)^2$$

- Apply it to solve the dual problem : replace the dot product by its square

SVC versus Logistic Regression

- SVC works well with unstructured and semi-structured data like test and images. Logistic regression works with already identified independent variables.
- SVC is based on geometrical properties of the data while logistic regression is based on statistical approaches
- The risk of overfitting is less in SVC
- General rule : try out logistic regression first

SVC versus Logistic Regression

n = number of features,
m = number of training examples

1. If *n is large (1–10,000) and m is small (10–1000)* : use logistic regression or SVM with a linear kernel.

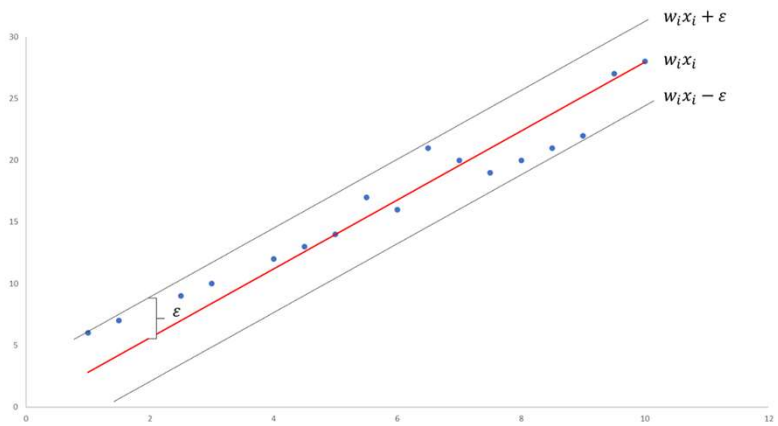
2. If *n is small (1–10 00) and m is intermediate (10–10,000)* : use SVM with (Gaussian, polynomial etc) kernel

3. If *n is small (1–10 00), m is large* (50,000–1,000,000+): first, manually add more features and then use logistic regression or SVM with a linear kernel

https://medium.com/axum-labs/logistic-regression-vs-support-vector-machines-svm-c335610a3d16#:~:text=SVM%20tries%20to%20finds%20the,are%20near%20the%20optimal%20point.

SVR

- Trick is to reverse the objective: try to fit as many instances as possible *on* the street while limiting margin violations
- Hyperparameter ε to control the width of the street called error margin



$$MIN \sum_{i=1}^{n} (y_i - w_i x_i)^2$$

$$|y_i - w_i x_i| \leq \varepsilon$$

$$MIN \frac{1}{2} ||\boldsymbol{w}||^2$$

Slack

- For any value that falls outside of ϵ, we can denote its deviation from the margin as ξ.

$$MIN\ \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{n}|\xi_i| \qquad |y_i - w_i x_i| \leq \varepsilon + |\xi_i|$$

  - Hyperparameter C :
    - If C increases: tolerance for points outside of ϵ increases.
    - If *C* approaches 0: tolerance → 0

- Two levels of tolerance to errors :
  - acceptable error margin ϵ
  - tuning the tolerance ξ of falling outside that acceptable error rate

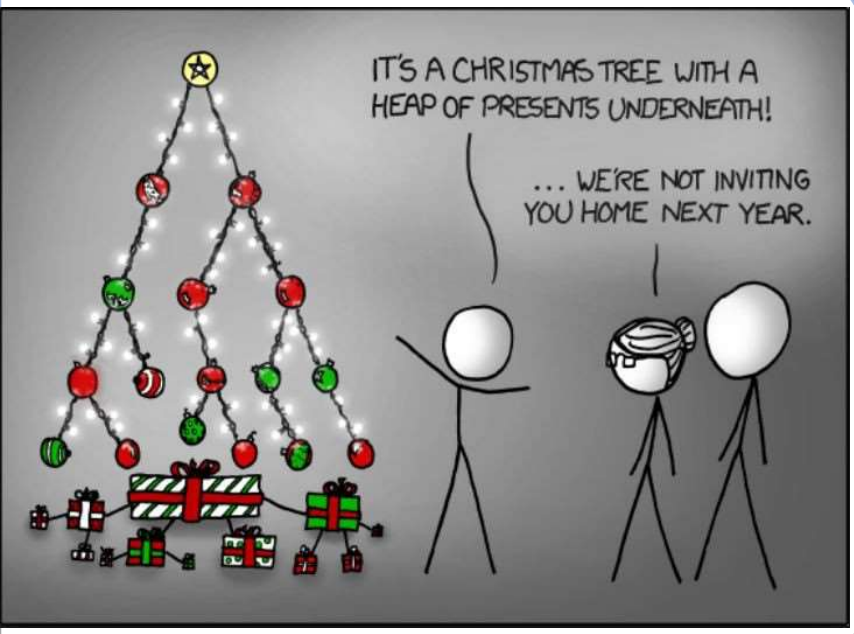
SVR in practice

- sklearn.svm.SRV :
  - free parameters : C and epsilon
  - fit time : scales at least quadratically with the number of samples and may be impractical beyond 10000 samples

```
>>> from sklearn.svm import SVR
>>> import numpy as np
>>> n_samples, n_features = 10, 5
>>> rng = np.random.RandomState(0)
>>> y = rng.randn(n_samples)
>>> X = rng.randn(n_samples, n_features)
>>> clf = SVR(C=1.0, epsilon=0.2)
>>> clf.fit(X, y)
SVR(epsilon=0.2)
```

- sklearn.svm.LinearSVR :
  - Similar to SVR with kernel='linear', but use of another library
  - Scale better to large number of samples (up to 100000)

```
>>> from sklearn.svm import LinearSVR
>>> from sklearn.datasets import make_regression
>>> X, y = make_regression(n_features=4, random_state=0)
>>> regr = LinearSVR(random_state=0, tol=1e-5)
>>> regr.fit(X, y)
LinearSVR(random_state=0, tol=1e-05)
>>> print(regr.coef_)
[16.35... 26.91... 42.30... 60.47...]
>>> print(regr.intercept_)
[-4.29...]
>>> print(regr.predict([[0, 0, 0, 0]]))
[-4.29...]
```

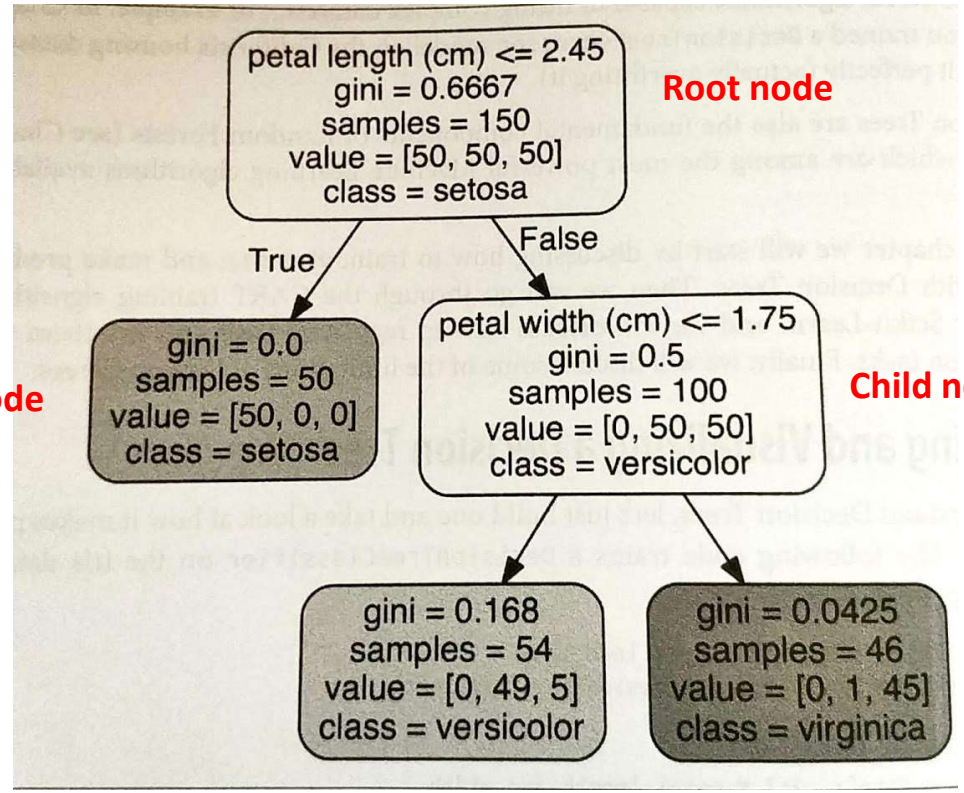


Ensemble Methods

Decision Trees

- Fundamental components of Random Forests

Classification example :



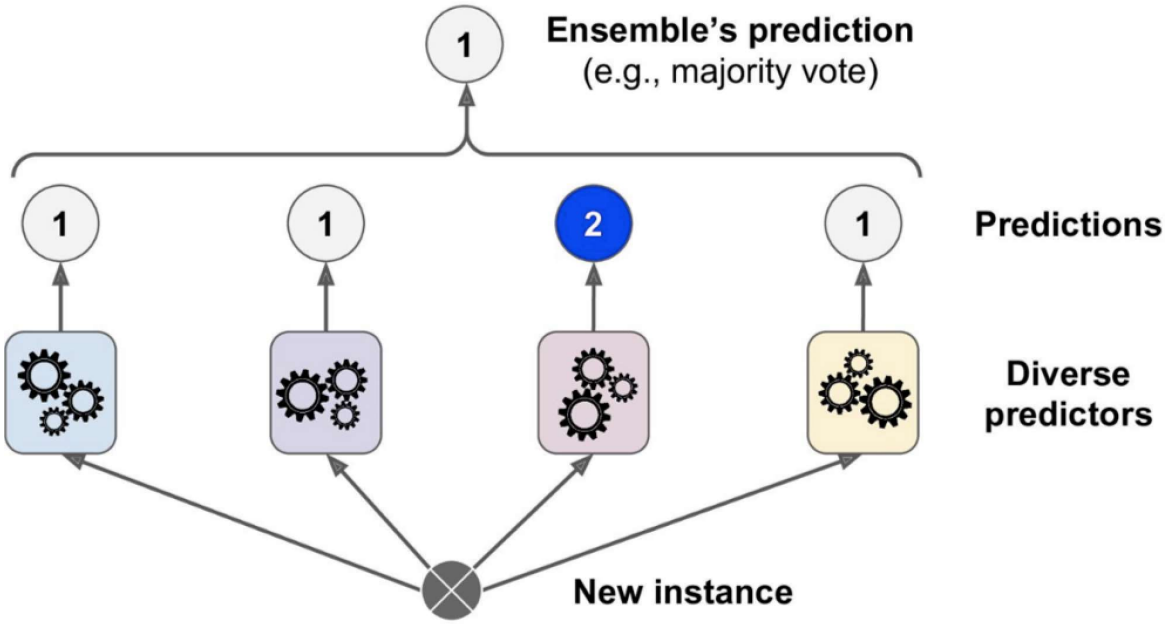
What would change for a regression example ?

- Regularization : maximum depth of the tree

Ensemble Methods

- Decision Trees are very sensitive to small variations in the training data.

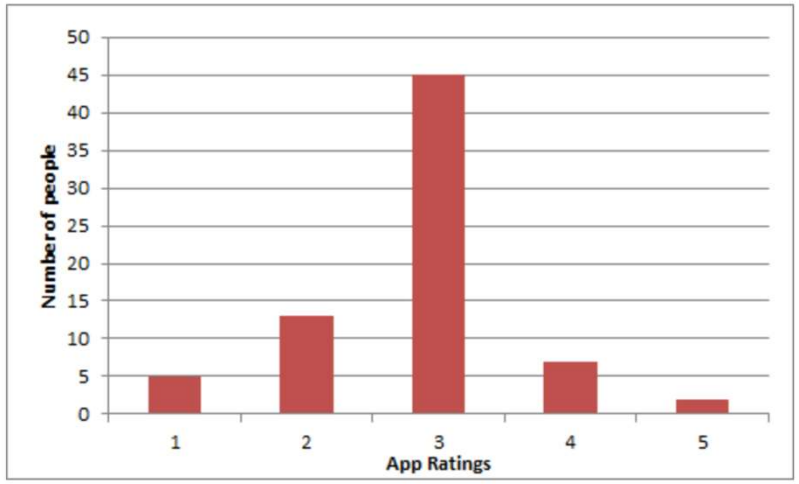
- *Wisdom of the crowd :* aggregate predictions of a group of predictors → Ensemble methods



Types of Ensemble Techniques

- Simple ensemble techniques
  - Mode, average, weighted average
- Advanced ensemble techniques
  - Bagging (Bootstrap AGGregatING)
  - Boosting

Simple Ensemble Techniques



Person	Professional	Weight	Rating
A	Y	0.3	3
B	Y	0.3	2
C	Y	0.3	2
D	N	0.15	4
E	N	0.15	3

1) Take the mode of the results

MODE=3, as majority people voted this

2) Take the average of the results (rounded to the nearest integer)

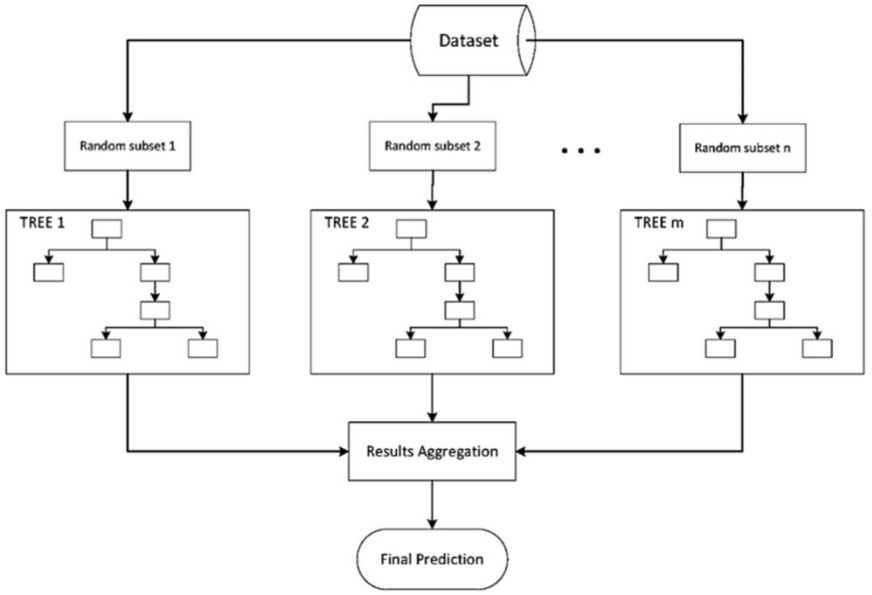
AVERAGE= (1*5)+(2*13)+(3*45)+(4*7)+(5*2)/72 = 2.833 = 3

3) Take the weighted average of the results

WEIGHTED AVERAGE= (0.3*3)+(0.3*2)+(0.3*2)+(0.15*4)+(0.15*3) =3.15 = 3

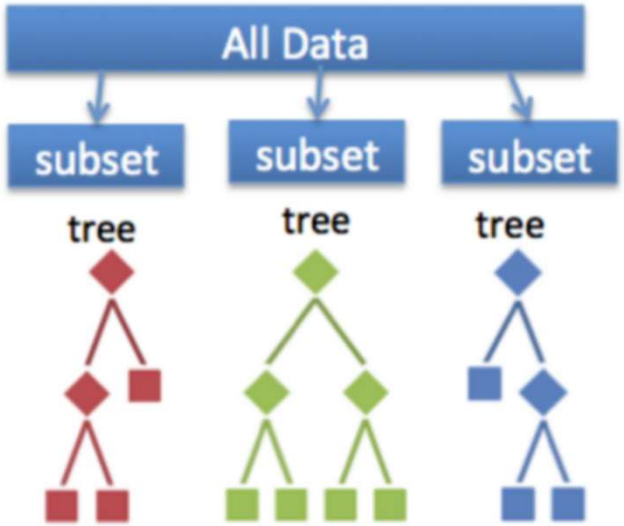
Bagging

- Use the same training algorithm for every predictor (e.g. classification tree)
- Train them on different random subsets of the training set (sampling *with* replacement)
- Combine using average or majority voting



Random Forest

- can be thought of as Bagging, with a slight tweak:

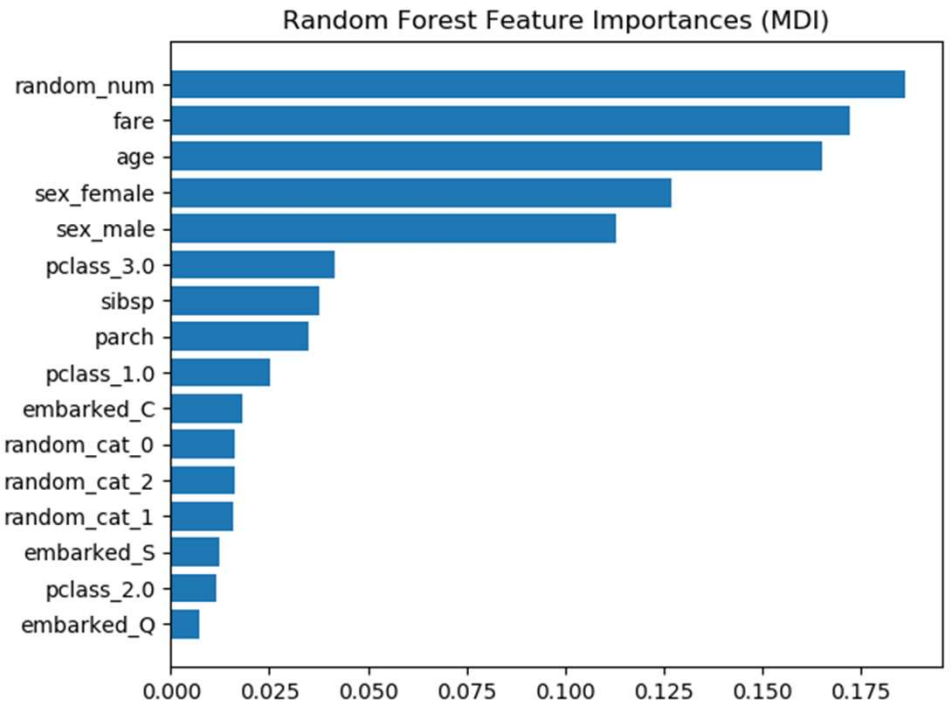


- Similar to bagging, bootstrapped subsamples are pulled from a larger dataset.
- The difference is that it searches for the best feature among a random subset of features

- Greater tree diversity, which trades a higher bias for a lower variance

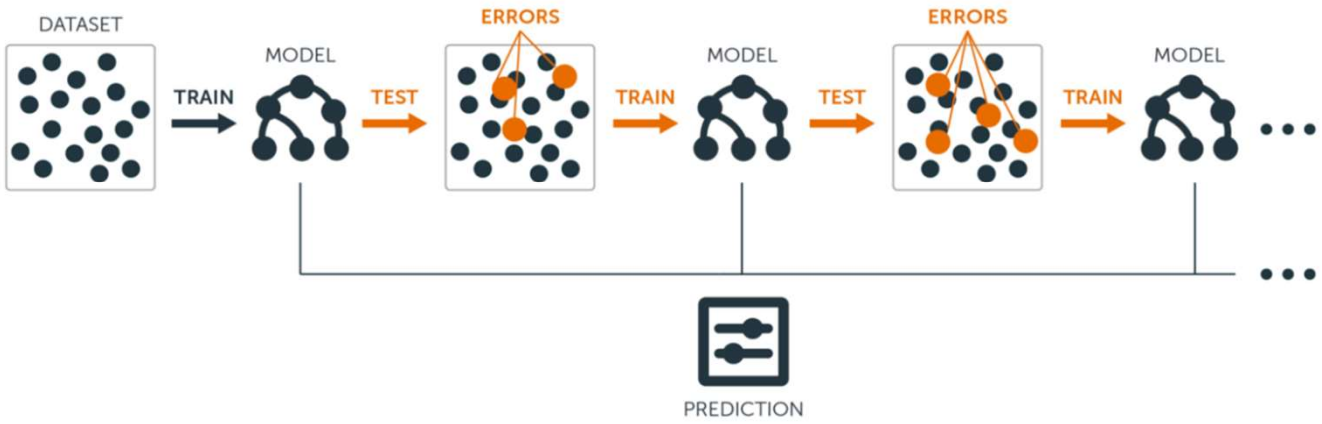
Feature Importance

- Relative importance of each feature, sum=1
- Useful for feature selection



Boosting

- Combine several weak learners into a strong learner : each model boots the performance of the ensemble
  - weak learners : each of them might not be good for the entire data set but is good for some part of the data set.



- Train predictors sequentially, each trying to correct the predecessor

- Examples of methods : AdaBoost and Gradient Boosting

Comparison

Which one is which ?

Similarities	• Uses voting • Combines models of the same type	
Differences	Individual models are built separately	Each new model is influenced by the performance of those built previously
	Equal weight is given to all models	Weights a model's contribution by its performance
Primary error reduction	Variance	Bias
Overfitting	Prevented, as each model only sees part of the data (helps decreasing the variance error)	Tends to overfit the training data (parameter tuning is crucial)

Ensemble Methods Pros and Cons

- Pros :
  - More accurate prediction results
    - better performance on unseen data as compared to the individual models in most of the cases
  - Stable and more robust
    - aggregate result of multiple models is always less noisy than the individual models
  - Used to capture the linear/non-linear relationships in data

- Cons :
  - Computation and design time is high
    - not good for real time applications
  - Selection of models for creating an ensemble is an Art !

Averaging methods in practice

	Classification	Regression

Bagging	`>>> from sklearn.ensemble import BaggingClassifier`
	`>>> from sklearn.neighbors import KNeighborsClassifier`
	`>>> bagging = BaggingClassifier(KNeighborsClassifier(),`
	`...  max_samples=0.5, max_features=0.5)`
		`>>> from sklearn.svm import SVR`
		`>>> from sklearn.ensemble import BaggingRegressor`
		`>>> from sklearn.datasets import make_regression`
		`>>> X, y = make_regression(n_samples=100, n_features=4,`
		`...  n_informative=2, n_targets=1,`
		`...  random_state=0, shuffle=False)`
		`>>> regr = BaggingRegressor(base_estimator=SVR(),`
		`...  n_estimators=10, random_state=0).fit(X, y)`
		`>>> regr.predict([[0, 0, 0, 0]])`
		`array([-2.8720...])`

RandomForest	`>>> from sklearn.ensemble import RandomForestClassifier`
	`>>> X = [[0, 0], [1, 1]]`
	`>>> Y = [0, 1]`
	`>>> clf = RandomForestClassifier(n_estimators=10)`
	`>>> clf = clf.fit(X, Y)`
		`>>> from sklearn.ensemble import RandomForestRegressor`
		`>>> from sklearn.datasets import make_regression`
		
		`>>> X, y = make_regression(n_features=4, n_informative=2,`
		`...  random_state=0, shuffle=False)`
		`>>> regr = RandomForestRegressor(max_depth=2, random_state=0)`
		`>>> regr.fit(X, y)`
		`RandomForestRegressor(max_depth=2, random_state=0)`
		`>>> print(regr.feature_importances_)`
		`[0.18146984 0.81473937 0.00145312 0.00233767]`
		`>>> print(regr.predict([[0, 0, 0, 0]]))`
		`[-8.32987858]`

Boosting methods in practice

* For >10000 samples : HistGradientBoosting is faster

AdaBoost

Classification

```
>>> from sklearn.model_selection import cross_val_score
>>> from sklearn.datasets import load_iris
>>> from sklearn.ensemble import AdaBoostClassifier

>>> X, y = load_iris(return_X_y=True)
>>> clf = AdaBoostClassifier(n_estimators=100)
>>> scores = cross_val_score(clf, X, y, cv=5)
>>> scores.mean()
0.9...
```

Regression

```
>>> from sklearn.ensemble import AdaBoostRegressor
>>> from sklearn.datasets import make_regression
>>> X, y = make_regression(n_features=4, n_informative=2,
...                        random_state=0, shuffle=False)
>>> regr = AdaBoostRegressor(random_state=0, n_estimators=100)
>>> regr.fit(X, y)
AdaBoostRegressor(n_estimators=100, random_state=0)
>>> regr.feature_importances_
array([0.2788..., 0.7109..., 0.0065..., 0.0036...])
>>> regr.predict([[0, 0, 0, 0]])
array([4.7972...])
>>> regr.score(X, y)
0.9771...
```

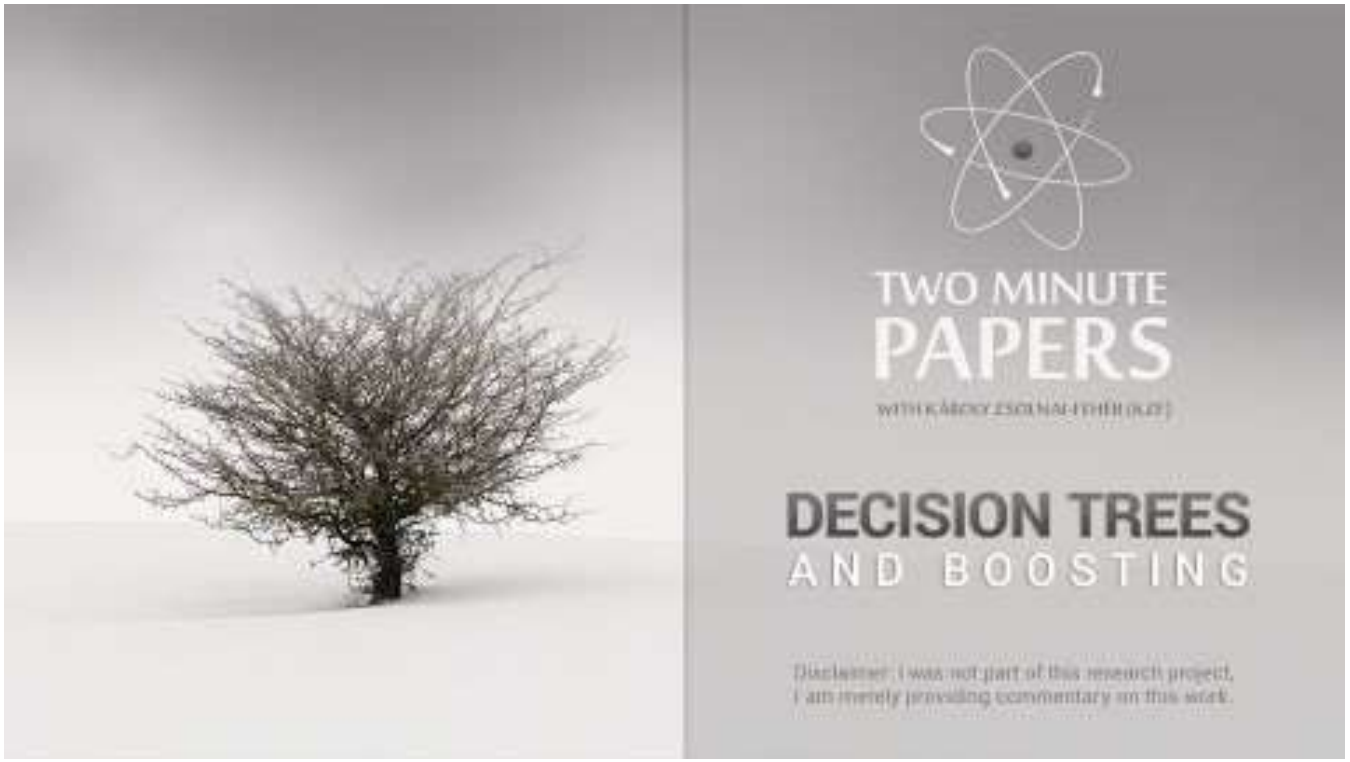
GradientBoosting*

Classification

```
>>> from sklearn.datasets import make_hastie_10_2
>>> from sklearn.ensemble import GradientBoostingClassifier

>>> X, y = make_hastie_10_2(random_state=0)
>>> X_train, X_test = X[:2000], X[2000:]
>>> y_train, y_test = y[:2000], y[2000:]

>>> clf = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,
...     max_depth=1, random_state=0).fit(X_train, y_train)
>>> clf.score(X_test, y_test)
0.913...
```

Regression

```
>>> import numpy as np
>>> from sklearn.metrics import mean_squared_error
>>> from sklearn.datasets import make_friedman1
>>> from sklearn.ensemble import GradientBoostingRegressor

>>> X, y = make_friedman1(n_samples=1200, random_state=0, noise=1.0)
>>> X_train, X_test = X[:200], X[200:]
>>> y_train, y_test = y[:200], y[200:]
>>> est = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1,
...     max_depth=1, random_state=0, loss='ls').fit(X_train, y_train)
>>> mean_squared_error(y_test, est.predict(X_test))
5.00...
```



Two-Minute Papers

Quiz

https://b.socrative.com/login/student/

Room : CONTI6128