Machine Learning
Lecture 2

Sardar Hamidian

The George Washington University
Department of Computer Science

Machine learning setup

Let us formalize the supervised machine learning setup. Our training data comes in pairs of inputs ($\mathbf{x}$, y), where $\mathbf{x} \in \boldsymbol{R}^{\mathbf{d}}$ is the input instance

and y its label. The entire training data is denoted as:

$$\mathbf{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subseteq \boldsymbol{R}^{\mathbf{d}} \times \boldsymbol{C}$$

where:

$\boldsymbol{R}^{\mathbf{d}}$:	d-dimensional feature space
$\mathbf{x}_i$:	input vector of the i^{th} sample
y_i:	label of the i^{th} sample
$\boldsymbol{C}$:	label space

Machine learning setup

- Binary classification, $\boldsymbol{C} = \{0, 1\}$ or $\boldsymbol{C} = \{-1, +1\}$
- Multi-class classification, $\boldsymbol{C} = \{1, 2, \cdots, K\}$ $(K \geq 2)$

Machine learning setup

- Binary classification, $\boldsymbol{C} = \{0, 1\}$ or $\boldsymbol{C} = \{-1, +1\}$
- Multi-class classification, $\boldsymbol{C} = \{1, 2, \cdots, K\}$ $(K \geq 2)$

} **Categorical**

Machine learning setup

- Binary classification, $\boldsymbol{C} = \{0, 1\}$ or $\boldsymbol{C} = \{-1, +1\}$
- Multi-class classification, $\boldsymbol{C} = \{1, 2, \cdots, K\}$ $(K \geq 2)$

} **Categorical**

- Regression, $\boldsymbol{C} = \mathbb{R}$ (Regression for predicting real numbers instead of distinct classes.)

Feature vectors

$\mathbf{x}_i^{\mathbf{d}}$: d dimensional feature vector of the i^{th} sample
$\mathbf{x}_i = \{ \boldsymbol{x^1}_i, \boldsymbol{x^2}_i, \boldsymbol{x^3}_i, \boldsymbol{x^4}_i, \ldots, \boldsymbol{x^{300}}_i \}$ **300** dimensions of the i^{th} sample.

Dense feature vectors:
A large number of the **300** features of the i^{th} sample are nonzero.

Sparse feature vectors:
A large number of the **300** features of the i^{th} sample are zero.

High dimensional feature vectors:
d = 10,000

Hypothesis classes

Hypothesis class: What function represents the relationship between the input vector and labels?

Hypothesis: Guess and make assumptions to come up with a hypothesis

No free lunch theorem: Accordingly no one ml method can be a good fit for all problems because each time there have to be assumptions.

Loss functions aka risk functions

Zero / one loss = error rate on this dataset *D.*

$$\mathcal{L}_{0/1}(h) = \frac{1}{n}\sum_{1}^{n} \delta_{h(\mathbf{x}_i)\neq y_i}, \text{ where } \delta_{h(\mathbf{x}_i)\neq y_i} = \begin{cases} 1, & \text{if } h(\mathbf{x}_i) \neq y_i \\ 0, & \text{o.w.} \end{cases}$$

Loss functions aka risk functions

Squared loss

If $|h(\mathbf{x}_i) - y_i|$ is a large difference, penalty gets magnified.

$$\mathcal{L}_{sq}(h) = \frac{1}{n}\sum_{i=1}^{n}(h(\mathbf{x}_i) - y_i)^2.$$

Loss functions aka risk functions

Absolute loss

Penalties don’t get magnified when $|h(\mathbf{x}_i) - y_i|$ is large .

$$\mathcal{L}_{abs}(h) = \frac{1}{n}\sum_{i=1}^{n} |h(\mathbf{x}_i) - y_i|$$

Generalization

Find function ***h*** that minimizes the loss

$$h = \text{argmin}_{h \in \mathcal{H}} \mathcal{L}(h)$$

Minimum loss

$$h(x) = \begin{cases} y_i, & \text{if } \exists(\mathbf{x}_i, y_i) \in D, \text{s.t.}, \mathbf{x} = \mathbf{x}_i, \\ 0, & \text{o.w.} \end{cases}$$

Loss would be zero

Overfitting ≈ no generalization

Train / Test splits

Training data:	learn from this	(%80 of data)
Validation data	check if you learned the patterns accurately	(%10 of data)
Test data	Evaluate the hypothesis	(%10 of data)

Train / Test splits

Training data:	learn from this	(%80 of data)
Validation data	check if you learned the patterns accurately	(%10 of data)
Test data	Evaluate the hypothesis	(%10 of data)

$$\textbf{Learning: } h^*(\cdot) = \textbf{argmin}_{h(\cdot)\in\mathcal{H}} \frac{1}{|D_{\text{TR}}|} \sum_{(\mathbf{x},y)\in D_{\text{TR}}} \ell(\mathbf{x}, y|h(\cdot))$$

Train / Test splits

Training data:	learn from this	(%80 of data)
Validation data	check if you learned the patterns accurately	(%10 of data)
Test data	Evaluate the hypothesis	(%10 of data)

$$\textbf{Validation: } \epsilon \text{VA} = \frac{1}{|D_{VA}|} \sum_{(\mathbf{x},y) \in D_{VA}} \ell(\mathbf{x}, y | h^*(\cdot))$$

= **should be small**

Train / Test splits

Training data:	learn from this	(%80 of data)
Validation data	check if you learned the patterns accurately	(%10 of data)
Test data	Evaluate the hypothesis	(%10 of data)

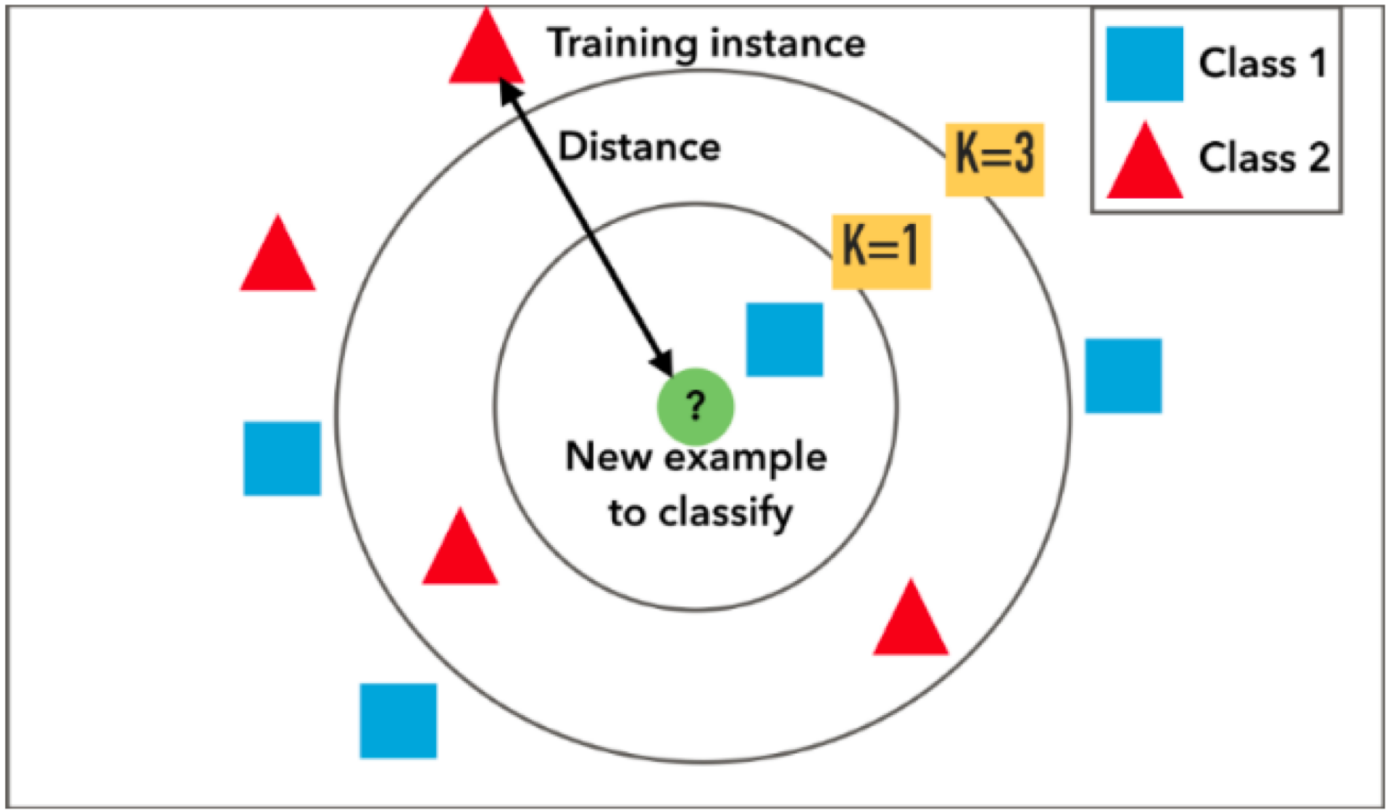
$$\textbf{Evaluation: } \epsilon_{\text{TE}} = \frac{1}{|D_{TE}|} \sum_{(\mathbf{x},y)\in D_{\text{TE}}} \ell(\mathbf{x}, y|h^*(\cdot))$$

Train / Test splits

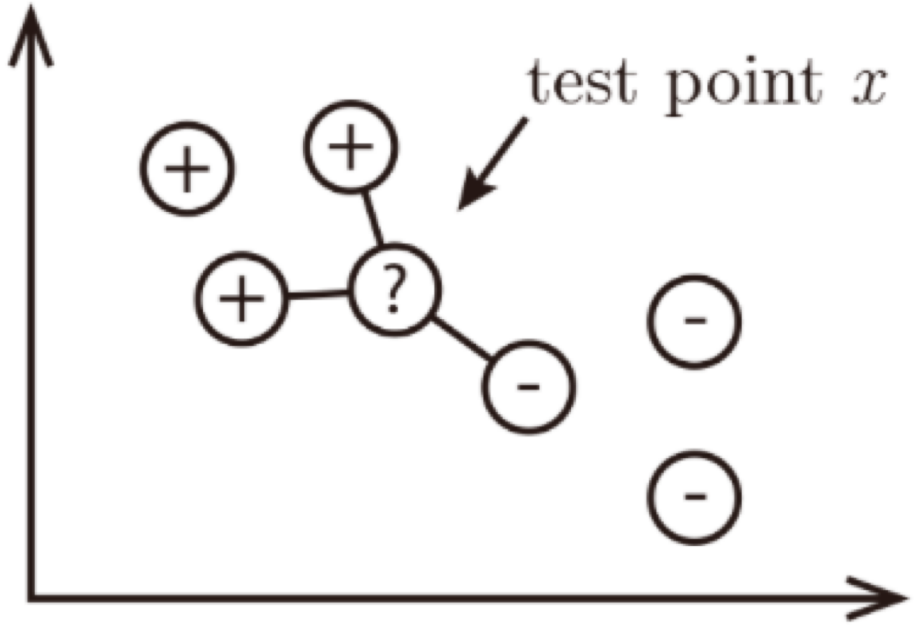
Training data:	learn from this	(%80 of data)
Validation data	check if you learned the patterns accurately	(%10 of data)
Test data	Evaluate the hypothesis	(%10 of data)

$$\textbf{Generalization: } \epsilon = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{P}}[\ell(\mathbf{x}, y|h^*(\cdot))]$$

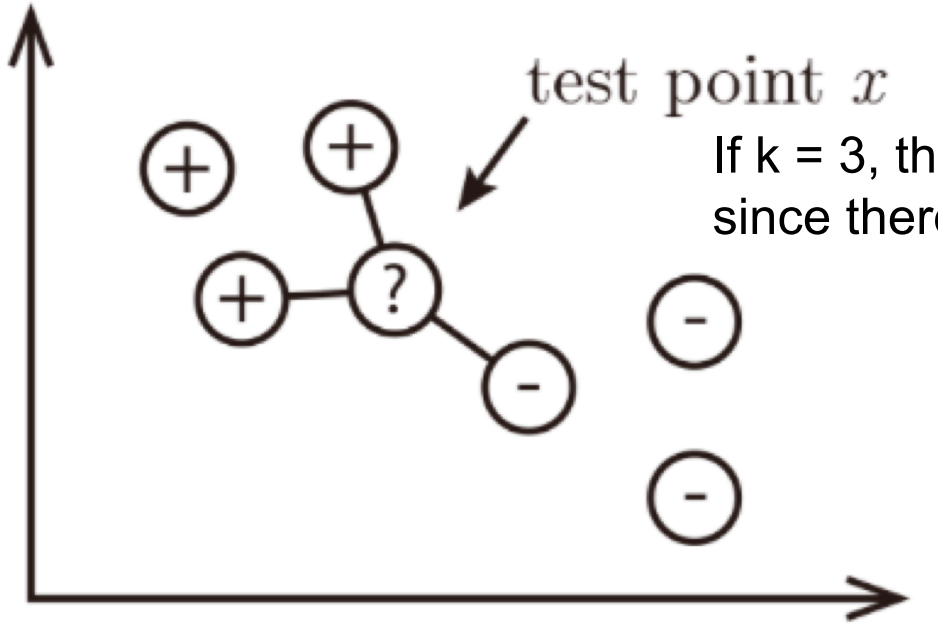
k-nearest neighbors aka KNN



k-nearest neighbors

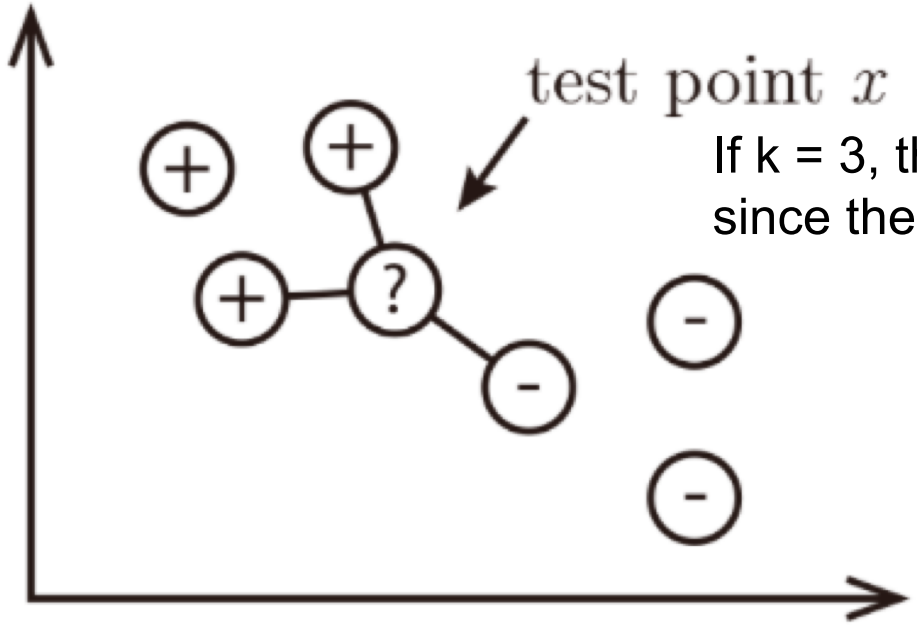


k-nearest neighbors



If k = 3, then x = ⊕
since there are (2 x ⊕) and only one ⊖

k-nearest neighbors



If k = 3, then x = $\oplus$
since there are (2 x $\oplus$) and only one $\ominus$

$$S_{\mathbf{x}} \subseteq D \text{ s.t. } |S_{\mathbf{x}}| = k$$

$$\forall (\mathbf{x}', y') \in D \backslash S_{\mathbf{x}},$$

$$\text{dist}(\mathbf{x}, \mathbf{x}') \geq \max_{(\mathbf{x}'', y'') \in S_{\mathbf{x}}} \text{dist}(\mathbf{x}, \mathbf{x}''),$$

$$h(\mathbf{x}) = \text{mode}(\{y'' : (\mathbf{x}'', y'') \in S_{\mathbf{x}}\}),$$

k-nearest neighbors aka KNN

- Non-parametric (no assumptions about underlying distribution)
  - Try knn when there is no prior information on data's distribution
- Lazy
  - Not eager
  - **No generalization** (no real training phase instead **instance based**)
- Simple
- Can be used for classification or regression
- K closest neighbors
  - Majority vote for classification
  - Mean or median for regression

k-nearest neighbors application areas

- Credit rating
- Loan decisions
- Binary vote classification
- Advanced
    - OCR (optical character recognition)
        - Handwriting recognition
    - Object recognition
    - Video recognition

KNN pros and cons

- Pros
  - Simple (easy to explain and interpret)
  - Versatile (both for classification and regression)
  - No assumption about data distributions
  - Relatively high accuracy (for such a simple method)
- Cons
  - Computationally expensive
  - Stores all training data - high memory requirement
  - Can be slow especially when ***n*** is large
  - Sensitive to irrelevant features and scale

Quiz#2: How does k affect the classifier? What happens if $k = n$? What if $k = 1$?

- $k = n$
- $k = 1$

Minkowski distance

$$\mathrm{dist}(\mathbf{x}, \mathbf{z}) = \left(\sum_{r=1}^{d} |x_r - z_r|^p \right)^{1/p}$$

- $p = 1$: |
- $p = 2$: |
- $p \to \infty$

Minkowski distance

$$\mathrm{dist}(\mathbf{x}, \mathbf{z}) = \left(\sum_{r=1}^{d} |x_r - z_r|^p \right)^{1/p}$$

- $p = 1$: Manhattan Distance (l_1-norm)
- $p = 2$: Euclidean Distance (l_2-norm)
- $p \to \infty$: Maximum Norm

The probabilistic approach (Supervised)

Y is set of classes.
X is data/features.

Example (of a fair and unfair coin).

$Y = \{Y_{FAIR}, Y_{CHEAT}\}$

$X = \{H, T\}$

P(X|Y) --- "Probability of X given Y"

Example: Probability of getting heads if you have a CHEATING coin.

P(Y|X) --- "Probability of Y given X"

Example: Probability of getting the CHEATING coin if you see heads

If you know these probabilities, you can make a classifier to determine which coin you have given just a bunch of flips. You might have learned this in a probability class. In a Machine Learning class, we need to first estimate these probabilities (somehow) from training data.

The entire training data is denoted as:

$$\mathbf{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subseteq \boldsymbol{R}^{\mathbf{d}} \times \boldsymbol{C}$$

In this case this would look like::

D = {

(H, CHEAT), (T, FAIR), (T, FAIR), (H,CHEAT),

(H,FAIR), (T,FAIR), (H,FAIR), (H,CHEAT) }

EXERCISE

Compute: P(X,Y), P(X|Y), P(Y|X), P(X), P(Y)

BAYES RULE derivation:

$$P(X,Y) = P(X)\,P(Y|X)$$

$$P(X,Y) = P(Y)\,P(X|Y), \text{ so}$$

$$P(X)\,P(Y|X) = P(Y)\,P(X|Y)$$

$$P(Y|X) = P(Y)\,P(X|Y) / P(X)$$

Probability of having cancer (**event**, Y) given that the person is a smoker (**evidence**, X)".
P(X) is called the **marginal likelihood**; this is the total probability of observing the evidence.
P(Y) is called the **prior**; this is the probability of our hypothesis without any additional prior information
P(X|Y) is called the **likelihood**; this is the probability of observing the new evidence, given our initial hypothesis
P(Y|X) is called the **posterior**; this is what we are trying to estimate.

The probabilistic approach

There are two cases of supervised learning:

- When we estimate P(Y|X) directly, then we call it *discriminative learning.*
    i. *Probability of the label given the data. The data helps to "discriminate" between different possible classes.*
- When we estimate P(X|Y)P(Y) or P(X,Y), then we call it *generative learning.*
    i. *Probability of the data given the label, or "how likely is an object of that class to generate the observed data".*

Example

`(1,0), (1,0), (2,0), (2, 1)`

`p(x,y)` is

```
      y=0   y=1
     -----------
x=1 | 1/2   0
x=2 | 1/4   1/4
```

`p(y|x)` is

```
      y=0   y=1
     -----------
x=1 | 1     0
x=2 | 1/2   1/2
```

Simple scenario: coin toss

Suppose you find a coin and it's ancient and very valuable. **Naturally**, you ask yourself, "What is the probability that it comes up heads when I toss it?" You toss it $n = 10$ times and get results: $H, T, T, H, H, H, T, T, T, T$. What is $P(H)$?

We observed n_H heads and n_T tails. So, intuitively,

$$P(H) \approx \frac{n_H}{n_H + n_T} = 0.4$$

Can we derive this formally?

Maximum Likelihood Estimation (MLE)

$$\theta_{MLE} = argmax_\theta \, P(D \, ; \, \theta)$$

For the sequence of coin flips we can use the **binomial distribution** to model $P(D \, ; \, \theta)$:

$$P(D \, ; \, \theta) = \binom{n_H + n_T}{n_H} \theta^{n_H} (1-\theta)^{n_T}$$

Now,

$$\begin{aligned}
\hat{\theta}_{MLE} &= argmax_\theta \binom{n_H + n_T}{n_H} \theta^{n_H} (1-\theta)^{n_T} \\
&= argmax_\theta \, log \binom{n_H + n_T}{n_H} + n_H \cdot log(\theta) + n_T \cdot log(1-\theta) \\
&= argmax_\theta \, n_H \cdot log(\theta) + n_T \cdot log(1-\theta)
\end{aligned}$$

We can now solve for θ by taking the derivative and equating it to zero. This results in

$$\frac{n_H}{\theta} = \frac{n_T}{1-\theta} \implies n_H - n_H\theta = n_T\theta \implies \theta = \frac{n_H}{n_H + n_T}$$

MLE highlights

- MLE gives the explanation of the data you observed.
- If n is large and your model/distribution is correct then MLE finds the **true** parameters.
- But the MLE can overfit the data if n is small.
- If you do not have the correct model (and n is small) then MLE can be terribly wrong!

For example, suppose you observe H,H. What is θ^{MLE}?

So what if you think you know something about θ?

Simple scenario: coin toss with prior knowledge

Assume you have a hunch that θ is close to $\theta' = 0.5$. But your sample size is small, so you don't trust your estimate.

Simple fix: Add m imaginery throws that would result in θ' (e.g. $\theta = 0.5$). Add m Heads and m Tails to your data.

$$\hat{\theta} = \frac{n_H + m}{n_H + n_T + 2m}$$

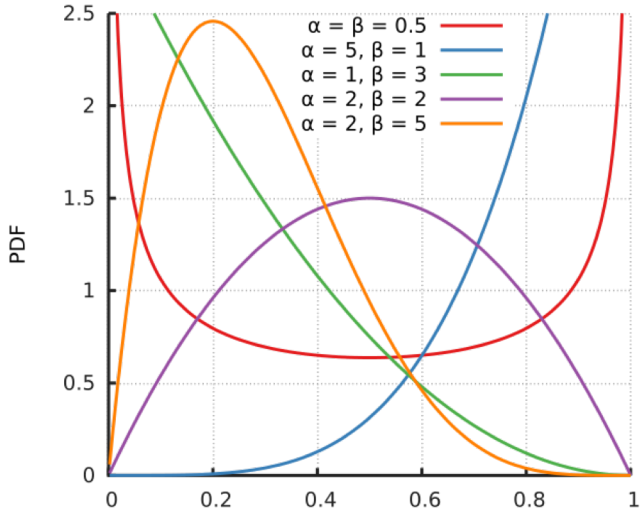
For large n, this is an insignificant change. For small n, it incorporates your "prior belief" about what θ should be.

The Bayesian Way

Model θ as a **random variable**, drawn from a distribution $P(\theta)$. Note that θ is **not** a random variable associated with an event in a sample space. In frequentist statistics, this is forbidden. In Bayesian statistics, this is allowed.

Now, we can look at $P(\theta \mid D) = \frac{P(D|\theta)P(\theta)}{P(D)}$ (recall Bayes Rule!), where

- $P(D \mid \theta)$ is the **likelihood** of the data given the parameter(s) θ,
- $P(\theta)$ is the **prior** distribution over the parameter(s) θ, and
- $P(\theta \mid D)$ is the **posterior** distribution over the parameter(s) θ.



Now, we can use the Beta distribution to model $P(\theta)$:

$$P(\theta) = \frac{\theta^{\alpha-1}(1-\theta)^{\beta-1}}{B(\alpha,\beta)}$$

where $B(\alpha,\beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$ is the normalization constant. Note that here we only need a distribution over a binary random variable. The multivariate generalization of the Beta distribution is the Dirichlet distribution.

Why use the Beta distribution?

- it models probabilities (θ lives on $[0, 1]$ and $\sum_i \theta_i = 1$)
- it is of the same distributional family as the binomial distribution (**conjugate prior**) $\rightarrow$ the math will turn out nicely:

$$P(\theta \mid D) \propto P(D \mid \theta) P(\theta) \propto \theta^{n_H + \alpha - 1} (1 - \theta)^{n_T + \beta - 1}$$

Note that in general θ are the parameters of our model. For the coin flipping scenario $\theta = P(H)$.
So far, we have a distribution over θ. How can we get an estimate for θ?

Maximum a Posteriori Probability Estimation (MAP)

For example, we can choose $\hat{\theta}$ to be the most likely θ given the data.

MAP Principle: Find $\hat{\theta}$ that maximizes the posterior distribution $P(\theta \mid D)$:

$$\begin{aligned}\hat{\theta}_{MAP} &= argmax_\theta \, P(\theta \mid D) \\ &= argmax_\theta \, logP(D \mid \theta) + logP(\theta)\end{aligned}$$

For out coin flipping scenario, we get:

$$P(\theta) = \frac{\theta^{\alpha-1}(1-\theta)^{\beta-1}}{B(\alpha,\beta)}$$

$$\begin{aligned}\hat{\theta}_{MAP} &= argmax_\theta \, P(\theta|Data) \\ &= argmax_\theta \, \frac{P(Data|\theta)P(\theta)}{P(Data)} \quad \text{(By Bayes rule)} \\ &= argmax_\theta \, log(P(Data|\theta)) + log(P(\theta)) \\ &= argmax_\theta \, n_H \cdot log(\theta) + n_T \cdot log(1-\theta) + (\alpha-1) \cdot log(\theta) + (\beta-1) \cdot log(1-\theta) \\ &= argmax_\theta \, (n_H + \alpha - 1) \cdot log(\theta) + (n_T + \beta - 1) \cdot log(1-\theta) \\ \implies \hat{\theta}_{MAP} &= \frac{n_H + \alpha - 1}{n_H + n_T + \beta + \alpha - 2}\end{aligned}$$

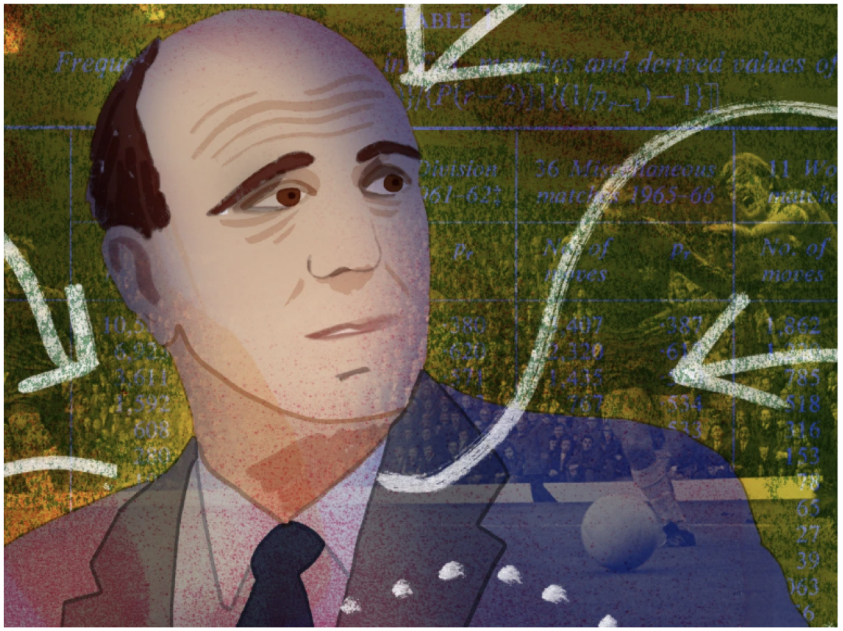
$$argmax_\theta \, n_H \cdot log(\theta) + n_T \cdot log(1-\theta)$$

- As $n \to \infty$, $\hat{\theta}_{MAP} \to \hat{\theta}_{MLE}$.
- MAP is a great estimator if prior belief exists and is accurate.
- If n is small, it can be very wrong if prior belief is wrong!

How One Man's Bad Math Helped Ruin Decades Of English Soccer

By Joe Sykes and Neil Paine

Filed under Ahead Of Their Time



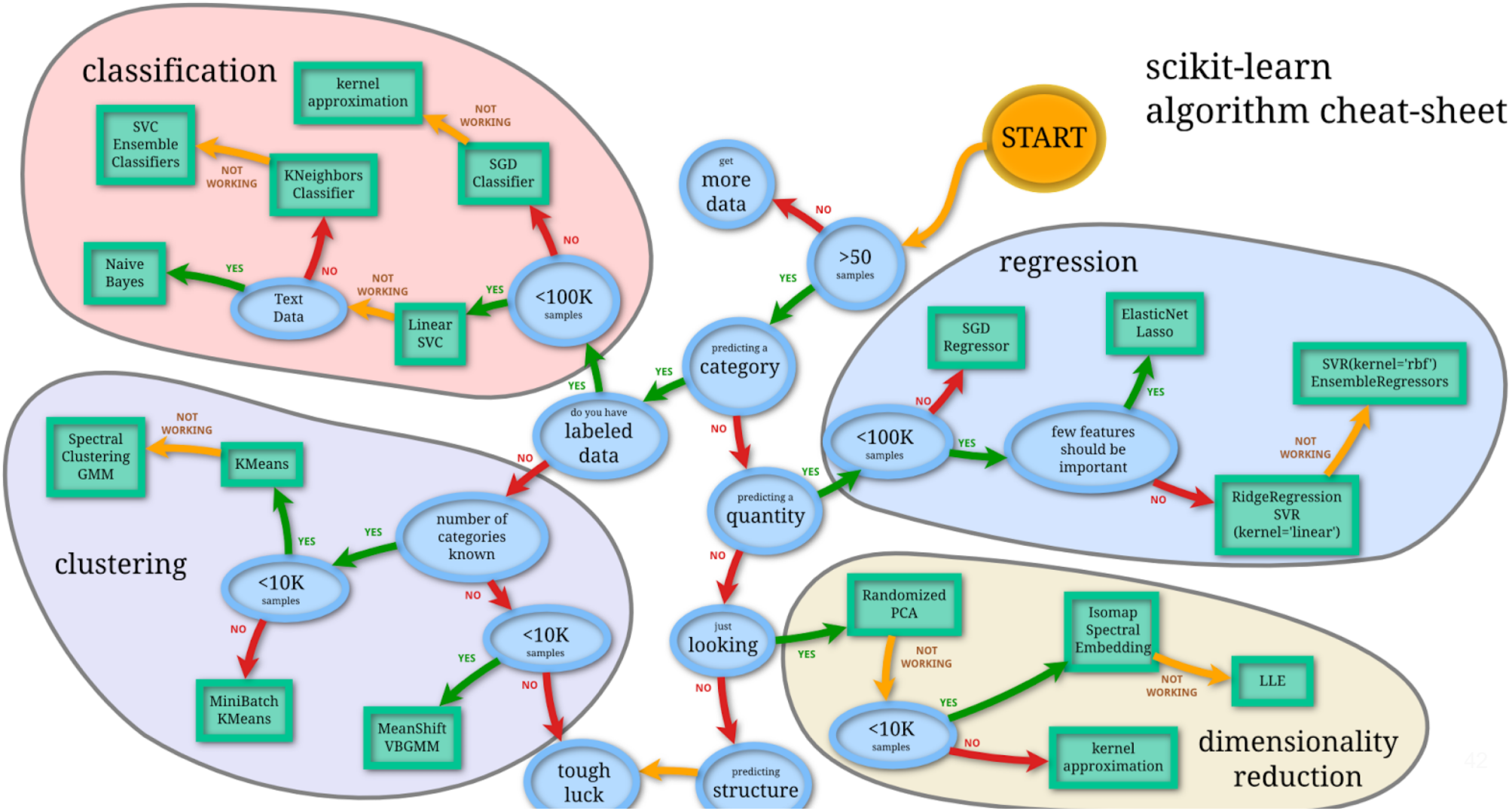
Here at FiveThirtyEight, we tend to think statistics can add to our understanding of sports. (What a surprise!) From the more mature sabermetric movements of baseball and basketball to growing ones in

Poring over all the scraps of data he'd collected, Reep eventually came to a realization: Most goals in soccer come off of plays that were preceded by three passes or fewer. And in Reep's mind, this basic truth of the game should dictate how teams play. The key to winning more matches seemed to be as simple as cutting down on your passing and possession time, and getting the ball downfield as quickly as possible instead. The long ball was Reep's secret weapon.

"Not more than three passes," Reep admonished during a 1993 interview with the BBC. "If a team tries to play football and keeps it down to not more than three passes, it will have a much higher chance of winning matches. Passing for the sake of passing can be disastrous."

The trouble was, Reep's theory was based on a fatally flawed premise. As I wrote two years ago, when discussing Reep's influence on soccer analytics

> Reep's mistake was to fixate on the percentage of goals generated by passing sequences of various lengths. Instead, he should have flipped things around, focusing on the probability that a given sequence would produce a goal. Yes, a large proportion of goals are generated on short possessions, but soccer is also fundamentally a game of short possessions and frequent turnovers. If you account for how often each sequence length occurs during the flow of play, *of course* more goals are going to come off of smaller sequences — after all, they're easily the most common type of sequence. But that doesn't mean a small sequence has a higher probability of leading to a goal.


scikit-learn
algorithm cheat-sheet
START
get more data
NO
>50 samples
YES
predicting a category
YES
do you have labeled data
YES
<100K samples
NO
Naive Bayes
YES
Text Data
NO
NOT WORKING
Linear SVC
YES
SVC Ensemble Classifiers
NOT WORKING
KNeighbors Classifier
NOT WORKING
kernel approximation
SGD Classifier
NO
classification
NO
predicting a quantity
YES
<100K samples
NO
SGD Regressor
YES
few features should be important
YES
ElasticNet Lasso
NO
RidgeRegression SVR (kernel='linear')
NOT WORKING
SVR(kernel='rbf') EnsembleRegressors
regression
NO
just looking
YES
Randomized PCA
NOT WORKING
<10K samples
YES
Isomap Spectral Embedding
NOT WORKING
LLE
NO
kernel approximation
dimensionality reduction
NO
predicting structure
tough luck
NO
number of categories known
YES
<10K samples
NO
MeanShift VBGMM
YES
<10K samples
NO
MiniBatch KMeans
YES
KMeans
NOT WORKING
Spectral Clustering GMM
clustering

Project 1 is coming