



**OPPA European Social Fund
Prague & EU: We invest in your future.**

Machine Learning and Data Analysis

Learning Logic Formulas

Filip Železný

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics
Intelligent Data Analysis lab
<http://ida.felk.cvut.cz>

January 6, 2012

PAC Learning

So far our PAC-learning framework considered *sample complexity*

- how fast m grows with $1/\epsilon$, $1/\delta$, and n
- we requested m to grow polynomially

Note about PAC-learning: inability to produce a consistent hypothesis implies inability to PAC-learn

- Fix a finite $X' \subseteq X$, set $P_X(x) = 1/|X'|$ for all $x \in X'$, set $\epsilon < \frac{1}{|X'|+1}$ and $\delta < 1$ (we are allowed to set any P_X , ϵ , and δ in PAC-learning).
- If hypothesis f is not consistent on an arbitrary example (x, y) , then $e(f) \geq 1/|X'| > \epsilon$, violating a PAC-learning condition with probability $1 > \delta$
- Thus if f is not consistent then we did not PAC-learn.

Efficient PAC-Learning

We now also consider *computational complexity*

Efficient PAC Learnability

An algorithm *efficiently PAC-learns* \mathcal{C} by \mathcal{F} if it PAC-learns \mathcal{C} by \mathcal{F} in polynomial time.

Polynomial: again in $1/\epsilon$, $1/\delta$, and the size n of examples

- Learning time grows at least as m does: learner needs at least a unit of time for processing each example
- Efficient PAC-learning thus requires each example to be processed in polynomial time
- Previous slide now implies: if finding a consistent model is NP-hard then we cannot efficiently PAC-learn (unless $RP=NP$)

Conjunctions and Disjunctions

$X = \{0,1\}^n$, i.e each $x = (x^1, \dots, x^n)$ where $x^i \in \{0,1\}$, $Y = \{0,1\}$

each f in $\mathcal{F} = \mathcal{C}$ defined by a conjunction ϕ of literals using propositional variables from set $\{p_1 \dots p_n\}$

$f(x) = 1$ iff ϕ is true under assignment of values x^i to p^i

Generalization algorithm:

$\phi = p_1 \wedge \neg p_1 \wedge \dots p_n \wedge \neg p_n$ {'most specific hypothesis'}

for each example $(x, 1) \in S$ **do**

for $i = 1 \dots n$ **do**

if $x^i = 0$ **then**

 delete p_i from ϕ

else

 delete $\neg p_i$ from ϕ

return ϕ

Conjunctions and Disjunctions (cont'd)

Algorithm never deletes a literal that must stay in ϕ . Final ϕ is thus consistent or no consistent ϕ exists.

A consistent algorithm exists and $|\mathcal{F}| = 3^n$, therefore conjunctions are PAC-learnable.¹

Sample complexity: $m \geq \frac{1}{\epsilon} \left(n \ln 3 + \ln \frac{1}{\delta} \right)$

Algorithm makes $m \cdot n$ steps, i.e. time linear in n (size of examples), therefore conjunctions are *efficiently PAC-learnable*.

Same applies for *disjunctions* using a simple transformation:

- run algorithm on 'negated' examples $(x, 1 - c(x))$
- negate its output ϕ ($\neg\phi$ is a disjunction)

¹ $|\mathcal{F}| = 2^{2n}$ if $p_i \wedge \neg p_i$ allowed in the conjunction.

k -Conjunctions and k -Disjunctions

Generalization algorithm produces the most specific (longest) consistent ϕ . Often, small ϕ are wanted.

A k -conjunction contains at most k literals. $\mathcal{C}^{k\text{conj}}$ is efficiently PAC-learnable simply by trying the $\mathcal{O}(n^k)$ possible k -conjunctions on n variables.

Heuristic approaches such as best-first search may be employed to speed-up the search within the polynomial bound. Search would start from the empty conjunction, adding a single literal in each step. The heuristic function evaluating the current conjunction ϕ would e.g. be

$$h(\phi) = -|\{(x, 0) \in S \mid x \models \phi\}|$$

while all descendants of any ϕ such that $x \not\models \phi$ for some $(x, 1) \in S$ would be pruned.

k -disjunctions $\mathcal{C}^{k\text{-disj}}$: analogical case, reduce by negating examples and ϕ

k -term DNF and k -clause CNF

A k -term DNF formula: disjunction of at most k conjunctions ('terms').

Example of a 3-term DNF formula:

$$(\neg p_1 \wedge p_3) \vee (p_2 \wedge \neg p_3 \wedge p_4 \wedge \neg p_6) \vee p_2$$

A k -clause CNF formula: conjunction of at most k disjunctions ('clauses').

Example of a 3-clause CNF formula:

$$(p_1 \vee \neg p_3) \wedge (\neg p_2 \vee p_3 \vee \neg p_4 \vee p_6) \wedge \neg p_2$$

Learnability results for the two classes analogical (again reduction by negation), we continue analysis with k -term DNF.

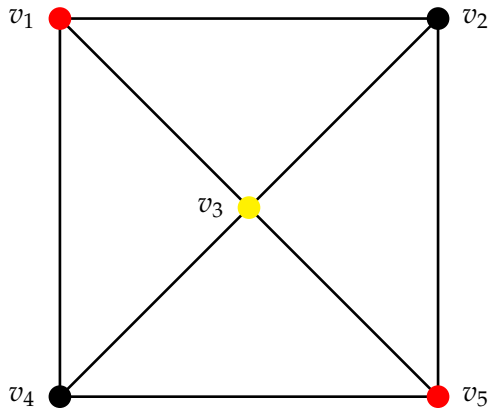
Consistent 3-term DNF as Graph Coloring

Finding a 3-term DNF formula consistent with a sample is as hard graph 3-coloring.

Graph 3-coloring:

- given vertices V and edges E ,
- assign one of 3 colors to each vertex $v \in V$ so that no adjacent vertices have same color
- NP-complete problem

Graph Coloring



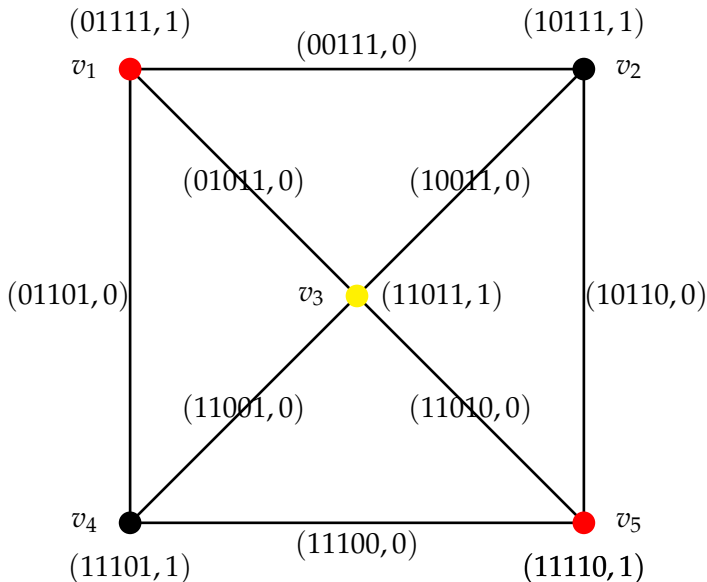
Reduction from a Graph to a Learning Sample

Graph	Sample
vertices $v_1 \dots v_n$	propositional variables $p_1 \dots p_n$
vertex v_i	example $(x, 1)$, $x^k = \begin{cases} 0 & \text{if } k = i \\ 1 & \text{otherwise} \end{cases}$
e.g.: vertex v_3	example $(11011, 1)$
edge e_{ij}	example $(x, 0)$, $x^k = \begin{cases} 0 & \text{if } k = i \text{ or } k = j \\ 1 & \text{otherwise} \end{cases}$
e.g.: edge v_{34}	example $(11001, 0)$

Reduction takes time linear in $m = |V| + |E|$ and n .

Remind: $(x, 1)$ denote positive examples, $(x, 0)$ negative examples.

Reduction from a Graph to a Learning Sample (cont'd)



Consistent 3-term DNF as Graph Coloring (cont'd)

Let S be a sample obtained by reduction of graph (V, E) . We will show:

- 1 If (V, E) is 3-colorable then there is a 3-term DNF formula ϕ consistent with S
- 2 If there is a 3-term DNF formula ϕ consistent with S then (V, E) is 3-colorable

Colorability \Rightarrow Consistency

Assume vertices V are split in partitions R, B, Y (red, black, yellow) representing a valid coloring.

Consider 3-term DNF formula

$$\phi = T_R \vee T_B \vee T_Y$$

such that

$$T_R = \bigwedge_{v_i \notin R} p_i$$

$$T_B = \bigwedge_{v_i \notin B} p_i$$

$$T_Y = \bigwedge_{v_i \notin Y} p_i$$

We will show that ϕ is consistent with S reduced from graph (V, E) .

Colorability \Rightarrow Consistency (cont'd)

Consistency with positive examples:

- 1 One positive example $(x, 1)$ for each vertex v_i
- 2 Assume $v_i \in R$ (B and Y are analogical)
- 3 T_R does not contain p_i (by definition of T_R)
- 4 $x^j = 1$ for $i \neq j$ (by reduction)
- 5 x satisfies T_R (denote $x \models T_R$) (from 3 and 4)
- 6 Therefore $x \models \phi$

Colorability \Rightarrow Consistency (cont'd)

Consistency with negative examples:

- 1 One negative example $(x, 0)$ for each edge e_{ij}
- 2 $x^i = 0$ (by definition)
- 3 v_i and v_j cannot both be red (because the coloring is valid)
- 4 Assume v_i is not red
- 5 $p_i \in T_R$ (by definition of T_R)
- 6 Therefore $x \notin T_R$ (from 2 and 5)
- 7 Analogically $x \notin T_B$ and $x \notin T_Y$ (repeat from Step 3 for the remaining colors)
- 8 Therefore $x \notin \phi$

Consistency \Rightarrow Colorability

Assume there is a consistent 3-term DNF ϕ , denote the 3 terms T_R, T_B, T_Y :

$$\phi = T_R \vee T_B \vee T_Y$$

This prescribes coloring:

for all positive examples $(x, 1)$ **do**

Let v_i be the vertex corresponding to x

if $x \models T_R$ **then**

color v_i red

else

if $x \models T_B$ **then**

color v_i black

else

if $x \models T_Y$ **then**

color v_i yellow

Consistency \Rightarrow Colorability (cont'd)

We prove that invalid coloring implies inconsistency of ϕ .

- 1 Suppose the coloring is not valid.
- 2 Then there are some adjacent v_i and v_j of same color, say red
- 3 Let $(x_i, 1)$, $(x_j, 1)$ and $(x_{ij}, 0)$ denote the examples corresponding to v_i , v_j and e_{ij}
- 4 $x_i, x_j \models T_R$ (by coloring algorithm)
- 5 $x_i^i = x_j^j = 0$ (by reduction)
- 6 T_R does not contain p_i or p_j (from 4 and 5)
- 7 $x_{ij}^k = 1$ for $k \notin \{i, j\}$ (by reduction)
- 8 $x_{ij} \models T_R$ (from 5 and 7)
- 9 Therefore $x_{ij} \models \phi$ but then ϕ is not consistent since $(x_{ij}, 0)$ is a negative example

3-term DNF not Efficiently PAC-Learnable

We proved that graph 3-coloring can be solved by linear-time reduction to a learning sample S and learning a 3-term DNF formula ϕ consistent with S .

Since graph 3-coloring is NP-hard, finding a consistent ϕ is also NP-hard.

Therefore $\mathcal{C}^{3\text{-term DNF}}$ is not efficiently PAC-learnable by $\mathcal{C}^{3\text{-term DNF}}$.

- This follows from the fact that inability to find a consistent hypothesis implies inability to PAC-learn (as we have already shown)

Can be also shown for any $\mathcal{C}^{k\text{-term DNF}}$, $k \geq 2$.

k -CNF and k -DNF

$\mathcal{C}^{k\text{-CNF}}$ contains conjunctions of k -disjunctions. Example:

$$(p_1 \vee p_2) \wedge (\neg p_3 \vee p_4 \vee p_5)$$

belongs in $\mathcal{C}^{3\text{-CNF}}$.

$\mathcal{C}^{3\text{-DNF}}$ analogical, we continue with $\mathcal{C}^{3\text{-CNF}}$.

$\mathcal{C}^{k\text{-CNF}}$ is as easy to learn as monotone conjunctions:

- assign a new atom p'_i to each clause that can be written with the original symbols p_i
- there is $\mathcal{O}(n^k)$ (i.e. poly number) of such clauses
- convert all examples into the new representation using symbols p'_i (in poly time)
- learn a monotone conjunction with the new examples using symbols p'_i
- convert it back to the original representation using symbols p_i

k -CNF vs. k -term DNF

Every k -term DNF formula can be written as an equivalent k -CNF formula.

Example:

$$(p_1 \wedge p_2) \vee (p_2 \wedge p_3) \equiv (p_1 \vee p_2) \wedge (p_1 \vee p_3) \wedge p_2 \wedge (p_2 \vee p_3)$$

Thus $\mathcal{C}^{k\text{-term DNF}} \subseteq \mathcal{C}^{k\text{-CNF}}$.

$$|\mathcal{C}^{k\text{-term DNF}}| = \mathcal{O}(2^n)$$

$$|\mathcal{C}^{k\text{-CNF}}| = \mathcal{O}\left(2^{\binom{2n}{k}}\right) = \mathcal{O}(2^{n^k})$$

So $\mathcal{C}^{k\text{-term DNF}} \subset \mathcal{C}^{k\text{-CNF}}$, thus not every k -CNF formula can be written as an equivalent k -term DNF formula.

Learning k -term DNF by k -CNF

Learning k -term DNF can be reduced to learning k -CNF. Assume examples in sample S contain values for n propositional variables.

- Create a new variable for each possible clause; there are $\mathcal{O}(n^k)$ of them
- Create a new sample S' using the new variables computed from the original variables.
- Learn a monotone conjunction from S' . Translating it back to the original variables yields a k -CNF formula

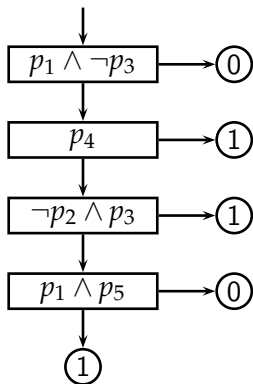
Since conjunctions are efficiently PAC-learnable, k -term DNF are efficiently PAC-learnable *by k -CNF*. (Caveat: Learning may produce a k -CNF formula not rewritable into a k -term DNF formula.)

In general: a hypothesis class may not be efficiently PAC-learnable by itself, but may be efficiently PAC-learnable by a larger hypothesis class!

k -Decision Lists

A k -Decision list is an ordered set of conjunctive rules with at most k literals in each, and a default value.

Example of a 2-DL:



k -Decision Lists (cont'd)

For $|\mathcal{C}^{k\text{-DL}}|$ we have

$$|\mathcal{C}^{k\text{-DL}}| = \mathcal{O}(3^{|\mathcal{C}^{k\text{-conj}}|} (|\mathcal{C}^{k\text{-conj}}|)!)$$

(each conjunction in the list can be either be absent, attached to 0, or 1, and the order in the list is arbitrary). Therefore $\log(|\mathcal{C}^{k\text{-DL}}|)$ is polynomial in n , implying polynomial sample complexity.

Every k -DNF formula can be written as a k -Decision List

- every term T of the formula (in any order) forms one rule $\boxed{T} \rightarrow 1$
- default value is 0

Thus

$$\mathcal{C}^{k\text{-DNF}} \subseteq \mathcal{C}^{k\text{-DL}}$$

For every $c \in \mathcal{C}^{k\text{-DL}}$, also $\neg c \in \mathcal{C}^{k\text{-DL}}$ (revert values in leaves). Therefore also

$$\mathcal{C}^{k\text{-CNF}} \subseteq \mathcal{C}^{k\text{-DL}}$$

k -Decision Lists (cont'd)

$\mathcal{C}^{k\text{-DL}}$ is efficiently PAC-learnable (by $\mathcal{C}^{k\text{-DL}}$) with the *covering algorithm*

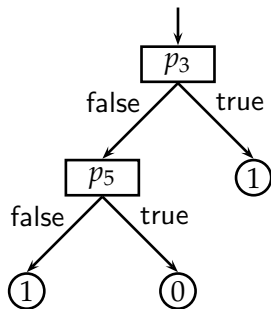
- 1: S = training sample, DL = empty decision list
- 2: **while** $S \neq \{\}$ **do**
- 3: ϕ = any k -conjunction such that
 $\{(x, 0) \in S \mid x \models \phi\} \neq \{\}$ and $\{(x, 1) \in S \mid x \models \phi\} = \{\}$ **or**
 $\{(x, 0) \in S \mid x \models \phi\} = \{\}$ and $\{(x, 1) \in S \mid x \models \phi\} \neq \{\}$
- 4: add $\boxed{\phi} \rightarrow 0$ **or** $\boxed{\phi} \rightarrow 1$ (respectively) to DL
- 5: $S = S \setminus \{(x, y) \in S \mid x \models \phi\}$
- 6: **if** $S = \{\}$ **then**
- 7: add default value 1 **or** 0 (respectively) to DL
- 8: **return** DL

Note: in Step 3 may go over all $\mathcal{O}(n^k)$ k -conjunctions; heuristic search applicable as in learning k -conjunctions.

k -Decision Trees

A tree in which each path from the root to a leaf has length at most k and represents a rule. Each non-leaf vertex contains one propositional variable, each leaf a class value.

Example of a 3-decision tree:



k -Decision Trees (cont'd)

Any k -DT can be represented by a k -DNF:

- create one term for each path leading to a leaf labelled with “1”

Any k -DT can be represented by a k -CNF:

- create one clause for each path leading to a leaf labelled with “0”

Therefore

$$\mathcal{C}^{k\text{-DT}} \subseteq \mathcal{C}^{k\text{-CNF}} \cap \mathcal{C}^{k\text{-DNF}}$$

Since $\mathcal{C}^{k\text{-CNF}} \neq \mathcal{C}^{k\text{-DNF}}$, we have $\mathcal{C}^{k\text{-DT}} \subset \mathcal{C}^{k\text{-CNF}}$ and $\mathcal{C}^{k\text{-DT}} \subset \mathcal{C}^{k\text{-DNF}}$ and since $\mathcal{C}^{k\text{-CNF}} \subseteq \mathcal{C}^{k\text{-DL}}$ we also have

$$\mathcal{C}^{k\text{-DT}} \subset \mathcal{C}^{k\text{-DL}}$$

k -Decision Trees (cont'd)

It is NP-hard to find a consistent k -Decision tree. $\mathcal{C}^{k\text{-DT}}$ is not efficiently PAC-learnable by $\mathcal{C}^{k\text{-DT}}$.

What is the error bound for an *inconsistent* tree? Remind: if

$$m \geq \frac{1}{2\epsilon^2} \ln \frac{2|\mathcal{F}|}{\delta}$$

then classification error will not exceed training error by more than ϵ with at least $1 - \delta$ probability.

Need to calculate $|\mathcal{F}| = |\mathcal{C}^{k\text{-DT}}|$

k -Decision Trees (cont'd)

$$|\mathcal{C}^{1\text{-DT}}| = 2$$

For depth $k + 1$ we have n choices of the root variable, $|\mathcal{C}^{k\text{-DT}}|$ possible left subtrees and $|\mathcal{C}^{(k\text{-DT})}$ possible right subtrees.

$$|\mathcal{C}^{(k+1)\text{-DT}}| = n \cdot |\mathcal{C}^{k\text{-DT}}|^2$$

Denote $l_k = \log_2 |\mathcal{C}^{k\text{-DT}}|$

$$l_1 = 1$$

$$l_{k+1} = \log_2 n + 2l_k$$

Solution:

$$l_k = (2^k - 1)(1 + \log_2 n) + 1$$

I.e. $\ln |\mathcal{C}^{k\text{-DT}}|$ polynomial in n (and exponential in k).

k -leave Decision Trees

Alternatively, we may bound the number of leaves.

$\mathcal{C}^{k\text{-leave DT}}$: trees with at most k leaves.

Finding a consistent k -leave DT still NP-hard. $\mathcal{C}^{k\text{-leave DT}}$ not efficiently PAC-learnable with $\mathcal{C}^{k\text{-leave DT}}$.

Error bound for an inconsistent tree? Size of the concept space:

$$|\mathcal{C}^{k\text{-leave DT}}| \leq n^{k-1}(k+1)^{(2k-1)}$$

Provides better bound than in k -DT: $\ln |\mathcal{C}^{k\text{-leave DT}}|$ polynomial in both n and k .

TDIDT algorithm

A recursive heuristic algorithm for quick (poly-time) construction of a possibly inconsistent DT .

TDIDT(S : sample, $P = \{p_1, \dots, p_n\}$: propositional variables)

if all examples in S have same class y **then**

return vertex labeled y

else

if $P = \{\}$ **then**

return vertex labeled by the *majority class* in S

else

Choose $p_i \in P$ and create a vertex labeled p_i

for $v \in \{0, 1\}$ **do**

 Create an edge from the p_i vertex, label it v

$S' = \{(x, y) \in S \mid x^i = v\}$

if $S' = \{\}$ **then**

 add a leaf to edge v , label it by the majority class in S

else

 add TDIDT($S', P \setminus p_i$) to edge v

TDIDT algorithm: remarks

- The heuristic in **Choose** $p_i \in P$

Define $S_i = \{(x, y) \mid x \models p_i\}$. Usually we choose p_i maximizing

$$\Delta H(S, p_i) = H(S) - \frac{|S_i|}{S} H(S_i) - \frac{|S \setminus S_i|}{S} H(S \setminus S_i)$$

where *entropy* $H(S)$ is defined as

$$H(S) = - \sum_{y \in \{0,1\}} \frac{|\{(x, y) \in S\}|}{|S|} \log_2 \frac{|\{(x, y) \in S\}|}{|S|}$$

Remarks

- TDIDT easily adaptable to constructing k -DT

Condition $P = \{\}$ is replaced by $P = \{\}$ or current depth = k

- TDIDT and other logic-based learners applicable also non-Boolean classification

TDIDT: No change in code needed. Decision lists: use multiple target values instead of 0 and 1, covering strategy remains same.

- TDIDT and other logic-based learners easily adaptable to nominal features

TDIDT: Instead of going over the Boolean range $v \in \{0,1\}$, we go over all possible values of the nominal feature x^i . Other learners: pre-construct Boolean features from nominal features (similarly to what follows).

Remarks (cont'd)

- TDIDT and other logic-based learners easily adaptable to real-valued features

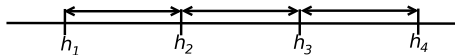
Use pre-constructed Boolean features such as p :

$$p \text{ is true iff } x^i > 153.56$$

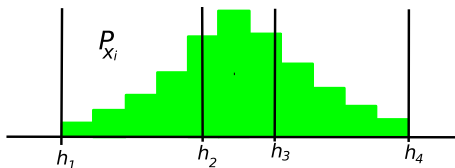
where x^i is an original real-valued feature and the threshold value 153.56 is determined in a preprocessing step. Multiple thresholds for one real-valued feature may be considered and used to define multiple Boolean features.

Discretization: 3 General Approaches

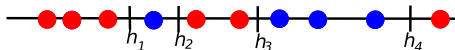
- Equilength intervals



- Equiprobable intervals



- Intervals containing same-class examples (most popular)



Inconsistent Hypotheses

Remind: when $\mathcal{C} \not\subseteq \mathcal{F}$ or $P_{Y|X}$ is not a concept, we must learn inconsistent hypotheses. Then we do not PAC-learn but we still have error bounds:

- Training error vs. classification error bound

$$|e(f) - e(S, f)| \leq \sqrt{\frac{1}{2m} \ln \frac{2|\mathcal{F}|}{\delta}}$$

does not assume the learner minimizes training error, i.e. that it outputs $\arg \min_{f \in \mathcal{F}} e(S, f)$

- Classification error of learned vs. best hypothesis bound

$$e(f) \leq \left(\min_{f \in \mathcal{F}} e(f) \right) + 2\sqrt{\frac{1}{2m} \ln \frac{2|\mathcal{F}|}{\delta}}$$

assumes the learner minimizes training error. This may be difficult.

Consistency vs. Error Minimization

Class	Find $f, e(S, f) = 0$	Find $\arg \min_{f \in \mathcal{F}} e(S, f)$
k -DT, k -leave DT	NP-hard	NP-hard
any \mathcal{C} where $ \mathcal{C} $ poly	easy	easy
.. such as k -conjunctions	easy	easy
general conjunctions	easy	NP-hard

Minimizing $e(S, f)$ for general conjunctions can be reduced to the NP-hard vertex-cover graph problem.



**OPPA European Social Fund
Prague & EU: We invest in your future.**
