Einführung in Active Learning

Matthias Richter & Maximilian Blanck

M3, 28. April 2021

Dr. Matthias Richter

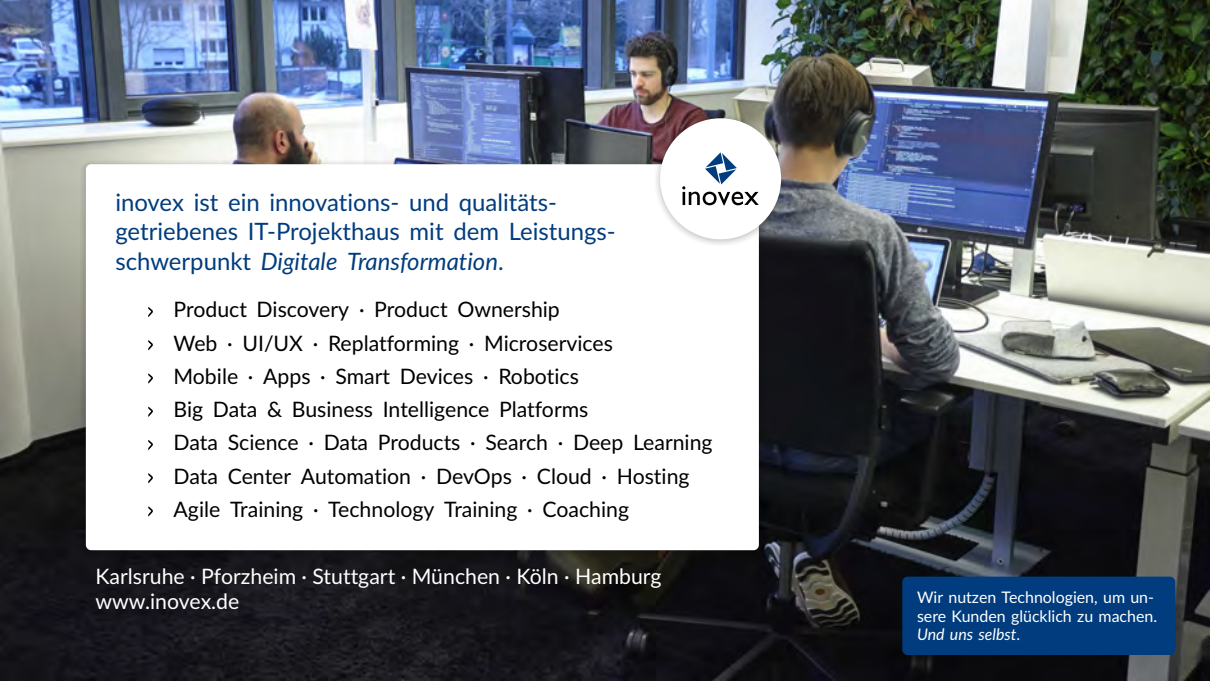
Machine Learning Engineer

matthias.richter@inovex.de

Maximilian Blanck

Data Scientist

maximilian.blanck@inovex.de



inovex

inovex ist ein innovations- und qualitätsgetriebenes IT-Projekthaus mit dem Leistungsschwerpunkt *Digitale Transformation*.

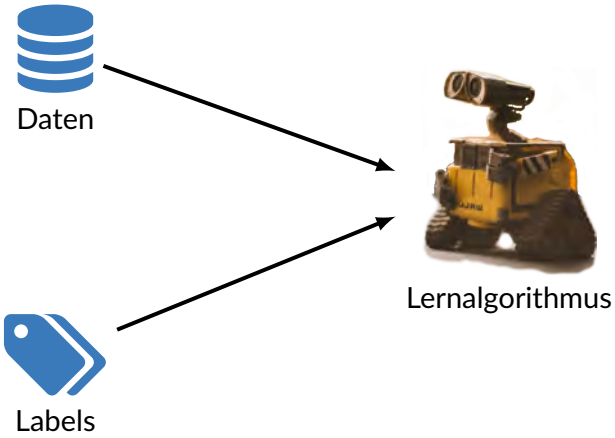
- Product Discovery · Product Ownership
- Web · UI/UX · Replatforming · Microservices
- Mobile · Apps · Smart Devices · Robotics
- Big Data & Business Intelligence Platforms
- Data Science · Data Products · Search · Deep Learning
- Data Center Automation · DevOps · Cloud · Hosting
- Agile Training · Technology Training · Coaching

Karlsruhe · Pforzheim · Stuttgart · München · Köln · Hamburg
www.inovex.de

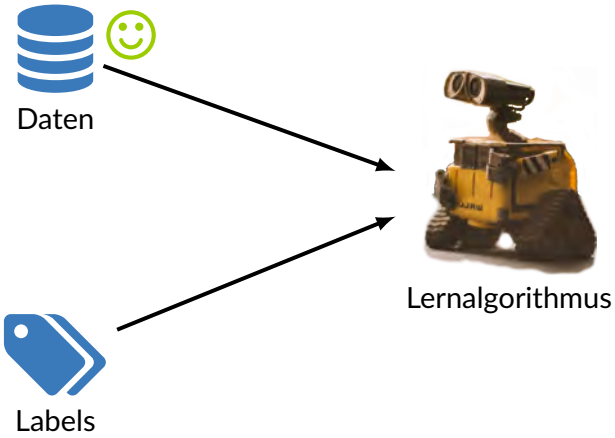
Wir nutzen Technologien, um unsere Kunden glücklich zu machen.
Und uns selbst.

Theorie

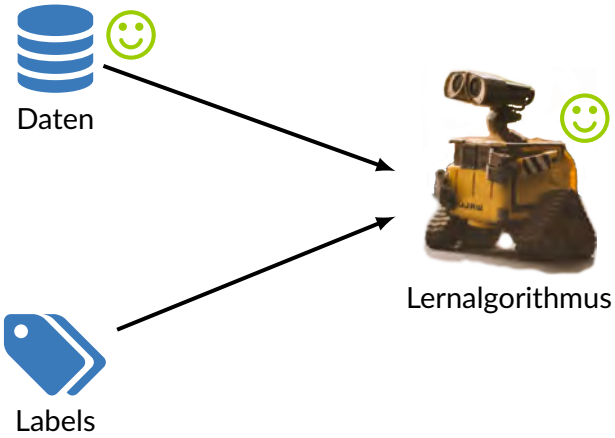
Drei Zutaten für (supervised) Machine Learning



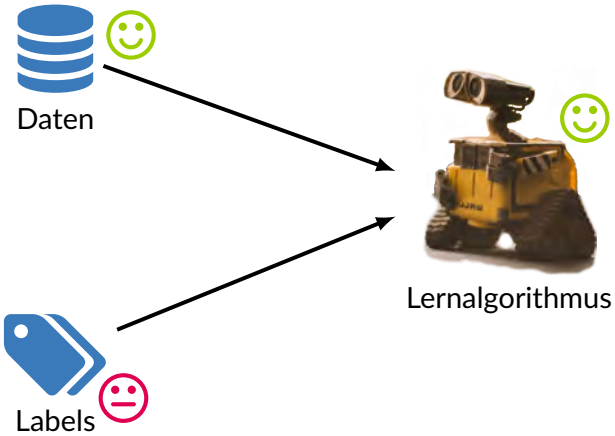
Drei Zutaten für (supervised) Machine Learning

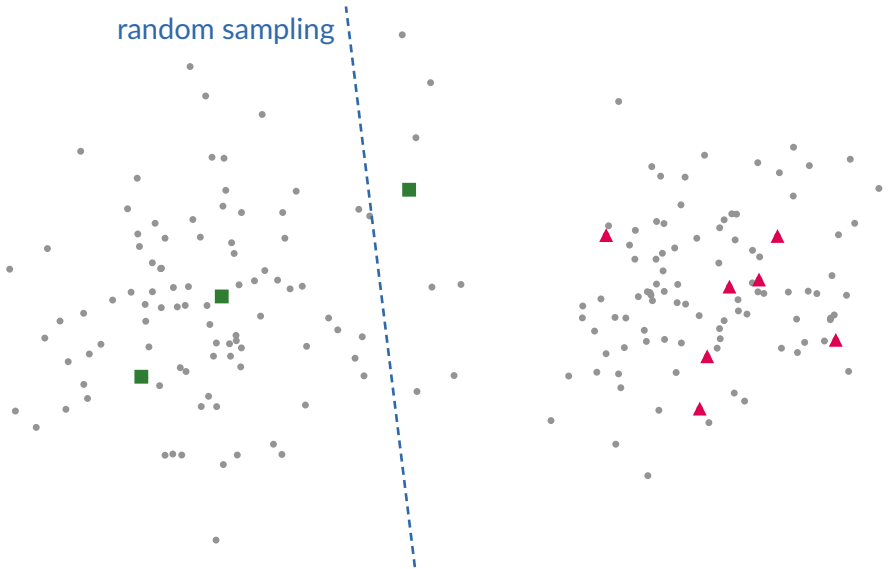


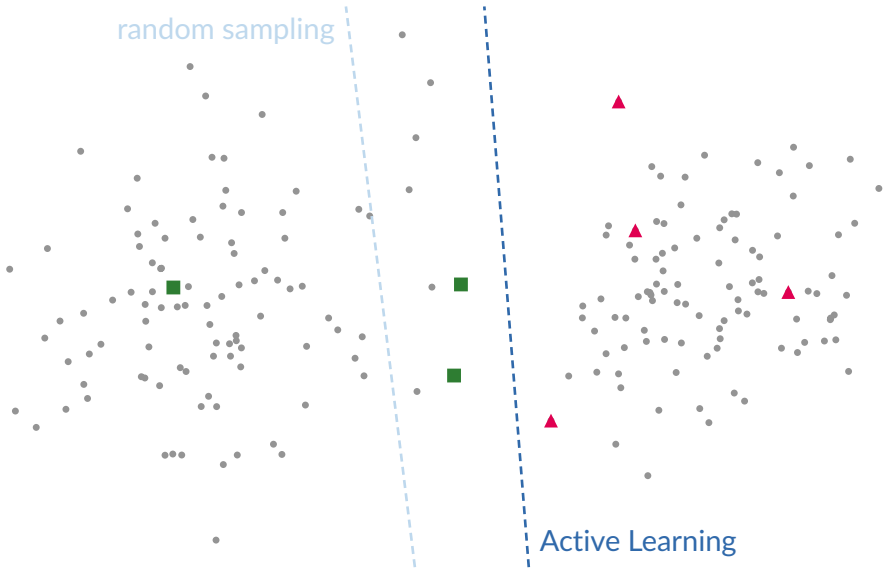
Drei Zutaten für (supervised) Machine Learning



Drei Zutaten für (supervised) Machine Learning




random sampling



inovex

Active Learning

Der Lernalgorithmus

- …nimmt aktiv am Training teil
- …erfragt Labels für interessante Daten
- …stellt so wenig Fragen wie möglich

aka *query learning*
aka *optimal experimental design*

Zutaten für Active Learning

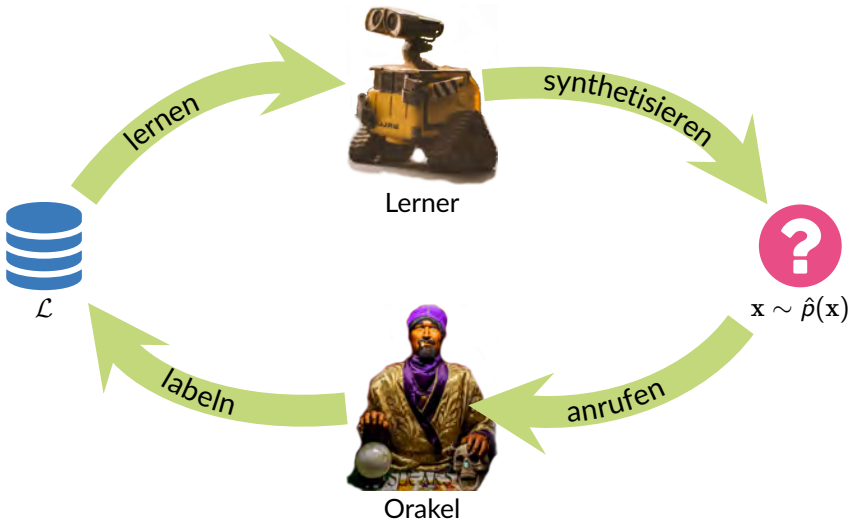
Gelabelte Daten
$\mathcal{L} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{L}$

Ungelabelte Daten
$\mathcal{U} = \{\mathbf{x}_k\}_{k=L+1}^{U}$

Lerner: Lernt Modell aus $\mathcal{L}$
$\hat{y} = h(\mathbf{x})$

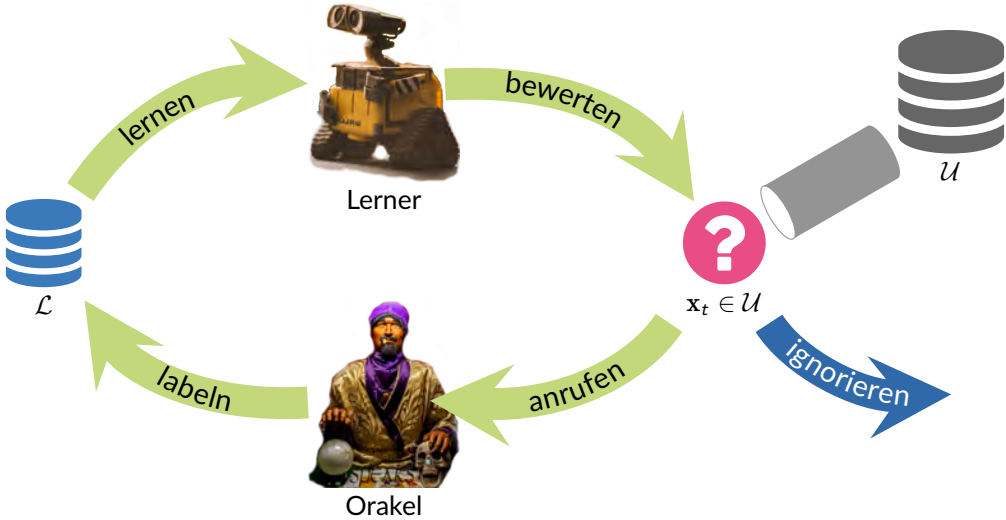
Orakel: Generiert Labels
$y = \omega(\mathbf{x})$

Query Synthesis



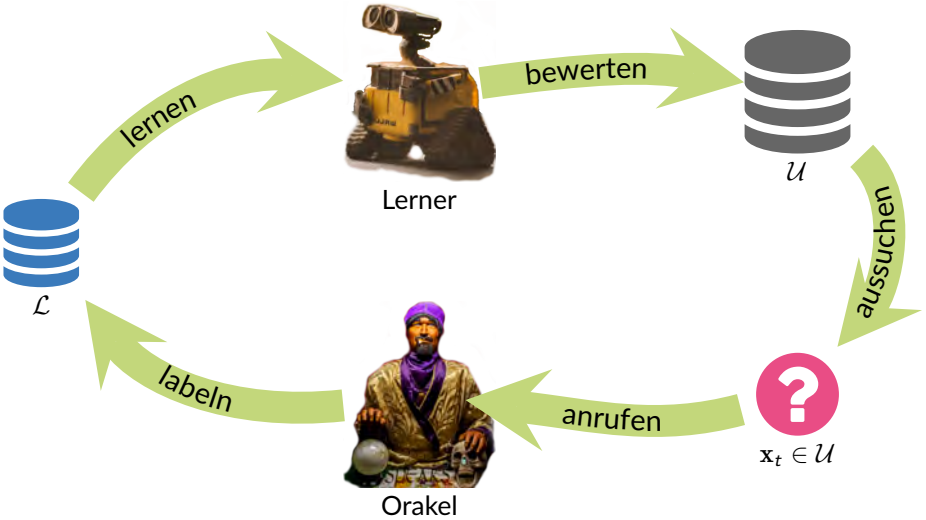
inovex

Selective Sampling



inovex

Pool-based Active Learning



inovex

Selective Sampling

$u(\mathbf{x}_t) > \tau \;\Rightarrow$ Orakel fragen

Pool-based Active Learning

$$\mathbf{x}_t = \arg\max_{\mathbf{x} \in \mathcal{U}} u(\mathbf{x})$$

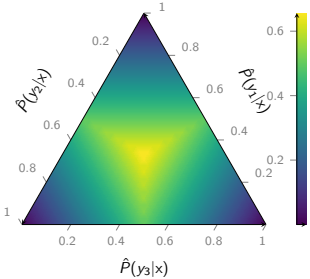
Uncertainty sampling

Nützlichkeit $\propto$ Unsicherheit des Modells

Uncertainty sampling

Nützlichkeit $\propto$ Unsicherheit des Modells

Least Confidence

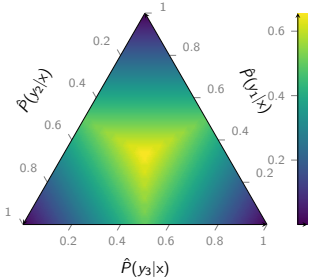


$$u(\mathbf{x}) = 1 - \max_y \hat{P}(y\,|\mathbf{x})$$

Uncertainty sampling

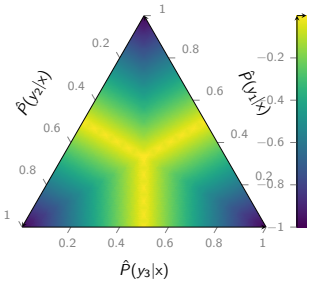
Nützlichkeit $\propto$ Unsicherheit des Modells

Least Confidence



$$u(\mathbf{x}) = 1 - \max_y \hat{P}(y\,|\mathbf{x})$$

Minimum Margin

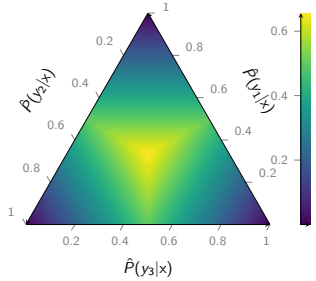


$$u(\mathbf{x}) = \hat{P}(y_2^\star\,|\mathbf{x}) - \hat{P}(y_1^\star\,|\mathbf{x})$$

Uncertainty sampling

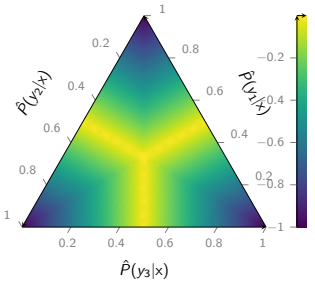
Nützlichkeit $\propto$ Unsicherheit des Modells

Least Confidence



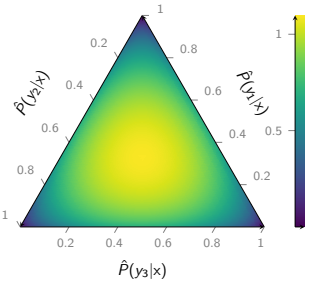
$$u(\mathbf{x}) = 1 - \max_y \hat{P}(y\,|\mathbf{x})$$

Minimum Margin



$$u(\mathbf{x}) = \hat{P}(y_2^\star\,|\mathbf{x}) - \hat{P}(y_1^\star\,|\mathbf{x})$$

Entropy

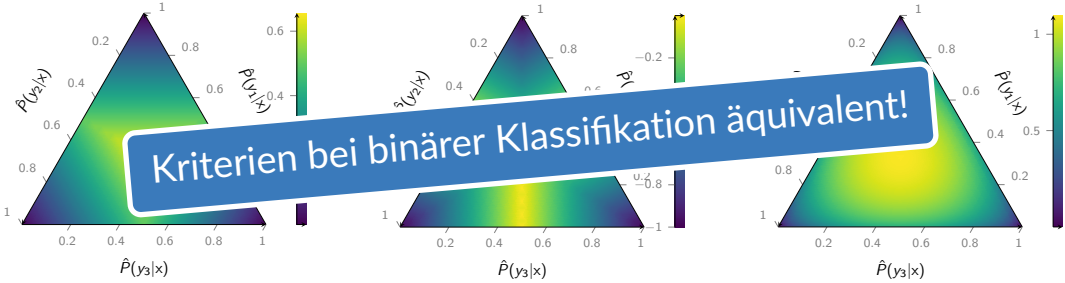


$$u(\mathbf{x}) = \mathbb{E}\left[-\log \hat{P}(y\,|\mathbf{x})\right]$$

Uncertainty sampling

Nützlichkeit $\propto$ Unsicherheit des Modells

Least Confidence | Minimum Margin | Entropy



Kriterien bei binärer Klassifikation äquivalent!

$$u(\mathbf{x}) = 1 - \max_y \hat{P}(y\,|\mathbf{x}) \qquad u(\mathbf{x}) = \hat{P}(y_2^\star\,|\mathbf{x}) - \hat{P}(y_1^\star\,|\mathbf{x}) \qquad u(\mathbf{x}) = \mathbb{E}\left[-\log \hat{P}(y\,|\mathbf{x})\right]$$

inovex

Query by Committee

- Problem: Uncertaintly Sampling verlässt sich auf *einen einzigen* Klassifikator
- Lösung: Mehrere Meinungen einholen

Committee $\mathcal{C} = \{h_1, \ldots, h_C\}$, trainiert auf $\mathcal{L}$

Nützlichkeit $\propto$ Uneinigkeit von $\mathcal{C}$

Uneinigkeitsmaße

Nützlichkeit $\propto$ Uneinigkeit von $\mathcal{C}$

- Vote Entropy:

$$u(\mathbf{x}) = -\sum_{y} \frac{V(y)}{C} \log \frac{V(y)}{C} \quad \text{mit} \quad V(y) = |\{h_i \mid h_i(\mathbf{x}) = y\}|$$

Uneinigkeitsmaße

Nützlichkeit $\propto$ Uneinigkeit von $\mathcal{C}$

- Vote Entropy:

$$u(\mathbf{x}) = -\sum_{y} \frac{V(y)}{C} \log \frac{V(y)}{C} \quad \text{mit} \quad V(y) = |\{h_i \mid h_i(\mathbf{x}) = y\}|$$

- Consensus Entropy:

$$u(\mathbf{x}) = -\sum_{y} \overline{P}(y|\mathbf{x}) \log \overline{P}(y|\mathbf{x}) \quad \text{mit} \quad \overline{P}(y|\mathbf{x}) = \sum_{h_i \in \mathcal{C}} \frac{\hat{P}_i(y|\mathbf{x})}{C}$$

Uneinigkeitsmaße

Nützlichkeit $\propto$ Uneinigkeit von $\mathcal{C}$

- Vote Entropy:

$$u(\mathbf{x}) = -\sum_{y} \frac{V(y)}{C} \log \frac{V(y)}{C} \quad \text{mit} \quad V(y) = |\{h_i \mid h_i(\mathbf{x}) = y\}|$$

- Consensus Entropy:

$$u(\mathbf{x}) = -\sum_{y} \overline{P}(y|\mathbf{x}) \log \overline{P}(y|\mathbf{x}) \quad \text{mit} \quad \overline{P}(y|\mathbf{x}) = \sum_{h_i \in \mathcal{C}} \frac{\hat{P}_i(y|\mathbf{x})}{C}$$

- Average KL-Divergence / Max Disagreement

$$u(\mathbf{x}) = \sum_{h_i \in \mathcal{C}} D_{\mathrm{KL}}(\hat{P}_i \| \overline{P}) \quad \text{mit} \quad D_{\mathrm{KL}}\left(\hat{P}_i \| \overline{P}\right) = \mathbb{E}_{y \sim \hat{P}_i}\left[\frac{\hat{P}_i(y|\mathbf{x})}{\overline{P}(y|\mathbf{x})}\right]$$

Expected Model Change

Nützlichkeit $\propto$ Erwartete Änderung des Modells

- Hängt vom Trainingsalgorithmus ab
- Gradient-Based $\rightsquigarrow$ Expected Gradient Length

$$
\begin{aligned}
u(\mathbf{x}) &= \mathbb{E}_y \left[\left\| \nabla \ell \Big(h; \mathcal{L} \cup \{(\mathbf{x}, y)\} \Big) \right\| \right] \\
&\approx \mathbb{E}_y \left[\left\| \nabla \ell \Big(h; \{(\mathbf{x}, y)\} \Big) \right\| \right]
\end{aligned}
$$

$\nabla \ell$ muss in jeder Runde für jedes Sample und für jedes mögliche Label berechnet werden!

Expected Error Reduction

Nützlichkeit $\propto$ Erwartete Minderung des Fehlers auf $\mathcal{U}$

- Nicht immer berechenbar
- Abhängig von Fehlermaß, z.B. mit 0/1-Loss:

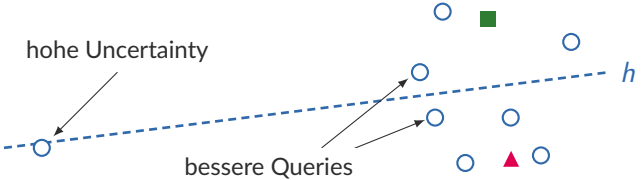
Trainiert auf $\mathcal{L} \cup \{\mathbf{x}, y\}$

$$u(\mathbf{x}) = \mathbb{E}_y \left[\sum_{\mathbf{x}_u \in \mathcal{U}} \left(\max_{y^*} \hat{P}'(y^* | \mathbf{x}_u) \right) \right]$$

Modell muss in jeder Runde für jedes Sample und für jedes mögliche Label neu trainiert und ausgewertet werden!

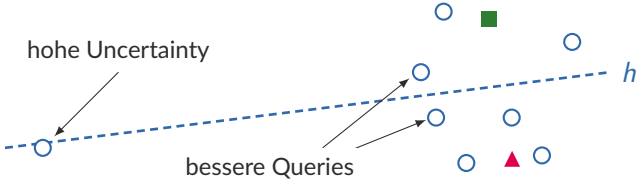
Linderung mit Variance Reduction, siehe (Burr Settles 2009)

Density Weighting



Bewertungsmaße beziehen sich nur auf ein $\mathbf{x} \in \mathcal{U}$

Density Weighting



Bewertungsmaße beziehen sich nur auf ein $\mathbf{x} \in \mathcal{U}$

$\rightsquigarrow$ Beziehe Informationen über die Dichte von $\mathcal{U}$ mit ein (Semi-Supervised Learning)

$$u'(\mathbf{x}) = u(\mathbf{x})g(\mathbf{x})^{\beta} \quad \text{z.B.} \quad g(\mathbf{x}) = \frac{1}{U}\sum_{\mathbf{x}' \in \mathcal{U}} \text{sim}(\mathbf{x}, \mathbf{x}')$$

Praxis

Und in der Praxis?! Sentiment Klassifikation von Tweets

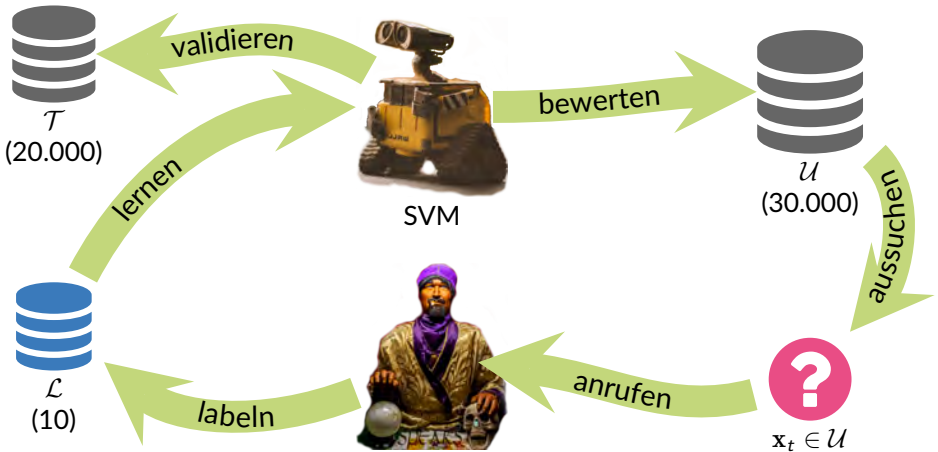
Wie ist die geäußerte Haltung/Stimmung in einem Tweet:

- 2 Klassen: [Negativ, Positiv]
- Gegeben 50.000 Tweets mit Annotationen
- Klassifikation mit Support Vector Machine und Tf-idf
  - Aktuelles Modell in Produktion: F1-Score (macro) = 0.95

Wie viele Annotationen hätten mit Active Learning eingespart werden können?

- Simulation mit 50.000 annotierten Tweets
- Pool-Set ($\mathcal{L} \cup \mathcal{U}$): 30.000
- Test-Set ($\mathcal{T}$): 20.000

Pool-based Active Learning



inovex

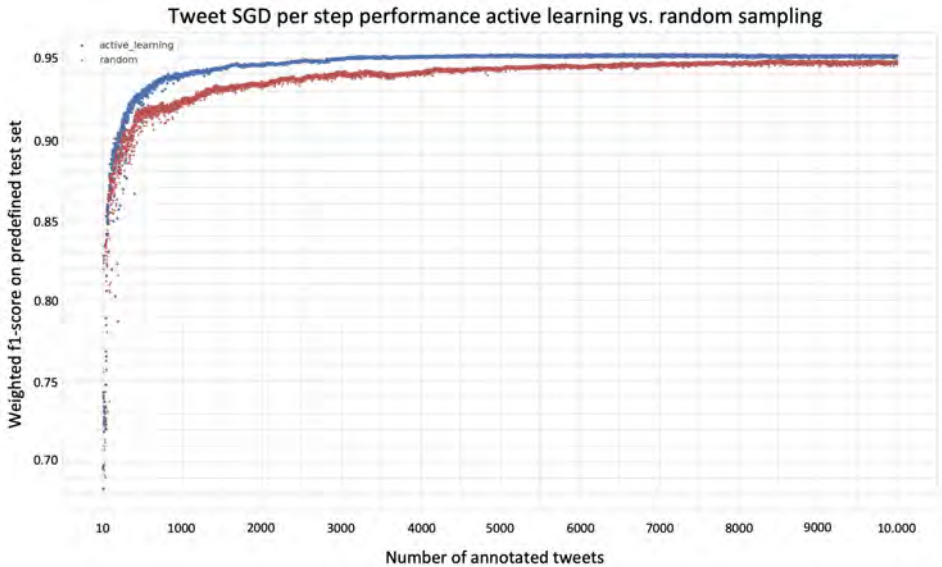
ModAL to the rescue

```
from modAL.models import ActiveLearner
from modAL.uncertainty import uncertainty_sampling
from sklearn.ensemble import RandomForestClassifier

learner = ActiveLearner(
        estimator=RandomForestClassifier(),
        query_strategy=uncertainty_sampling)

while not done:
    # sample instance to label next
    query_idx, x_t = learner.query(X)

    # query the label the oracle
    y_t = oracle(x_t)

    # supply label for queried instance
    learner.teach(x_t, y_t)
```

Klassifikation mit Support Vector Machine und Tf-idf

```
# split into train and test set
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=20000)
# split into initial pool and train set
X_pool, X_train, y_pool, y_train = train_test_split(X_train, y_train,
                                                    test_size=10)
# initialize model and ModAL learner
model_al = SGDCalibratorWrapper(SGDClassifier())
learner_al = ActiveLearner(
    estimator=model_al,
    query_strategy=uncertainty_sampling,
    X_training=X_train, y_training=y_train
)
# start active learning loop
while not done:
    query_idx, _ = learner_al.query(X_pool)
    learner_al.teach(X_pool[query_idx], y_pool[query_idx])
```

Active Learning Simulation



inovex

Zusammenfassung

Wie viele Annotationen hätten mit Active Learning eingespart werden können?

- Mehr als 50% der Annotationen können mit Active Learning eingespart werden.
- Active Learning wird nun für zukunftige Use-Cases (Klassifikation und Extraktion) eingesetzt.
- Query Batch Größe: 100 (ist Use-Case abhängig)

Praktische Tipps

- Mit bestehenden Daten eine Simulation starten.
- Test- und Validationset nicht mit Active Learning erstellen.
- Domänewissen ebenfalls mit einbringen, z.B. Density Weighting.
- Verschiedene Query Batch Größen ausprobieren.
- Komplettes Fitting (empfohlen) vs. Partielles Fitting.

No-free-Lunch!

- Was passiert wenn später das Modell ausgetauscht wird?
  (Lowell et al. 2009)

Ressourcen

- Burr Settles: Active Learning Literature Survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison. 2009.
- Burr Settles: Active Learning. Morgan & Claypool. 2012. ISBN: 9781608457267.
- Lowell, Lipton, Wallace: Practical Obstacles to Deploying Active Learning. 2009. aclweb.org/anthology/D19-1003.pdf
- ModAL Dokumentation: modal-python.readthedocs.io
- Demo Notebook: github.com/inovex/intro-to-active-learning

Dr. Matthias Richter
Machine Learing Engineer

inovex GmbH
Ludwig-Erhard-Allee 6
76131 Karlsruhe

matthias.richter@inovex.de
+49 1523 - 31 81 258

Maximilian Blanck
Data Scientist

inovex GmbH
Ludwig-Erhard-Allee 6
76131 Karlsruhe

maximilian.blanck@inovex.de
+49 1523 - 31 81 269

inovex

SGD Wrapper

```
from sklearn.base import BaseEstimator
from sklearn.calibration import CalibratedClassifierCV
class SGDCalibratorWrapper(BaseEstimator):
    def __init__(self, svm):
        self.svm = svm
        self.calibrator = CalibratedClassifierCV(self.svm, cv='prefit')

    def fit(self, X, y):
        self.svm.fit(X, y)
        self.calibrator.fit(X, y)
        return self

    def predict(self, X):
        return self.svm.predict(X)

    def predict_proba(self, X):
        return self.calibrator.predict_proba(X)
```