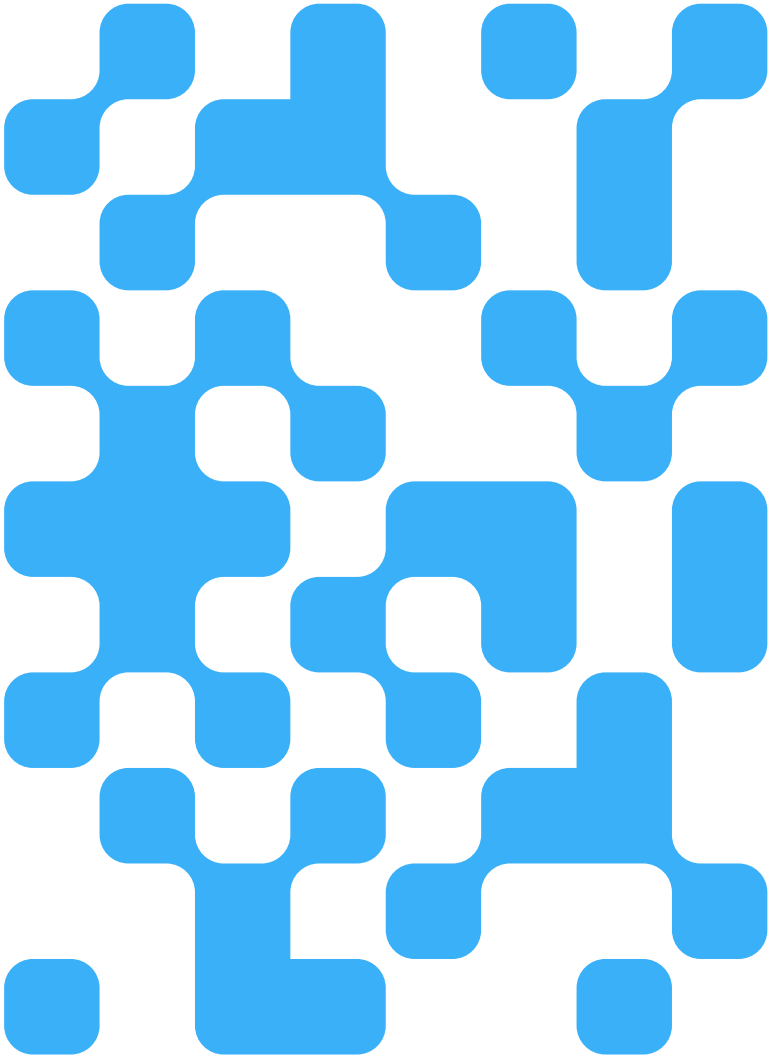
Machine Learning

2024 (ML-2024)

Lecture 11. Back to the grounds: Gradient Boosting, k-Nearest Neighbors, Support Vector Machine

by Alexei Valerievich Kornaev, Dr. habil. in Eng. Sc.,
Researcher at the RC for AI, Assoc. Prof. of the Robotics and CV Master's Program, Innopolis University
Researcher at the RC for AI, National RC for Oncology n.a. NN Blohin
Professor at the Dept. of Mechatronics, Mechanics, and Robotics, Orel State University

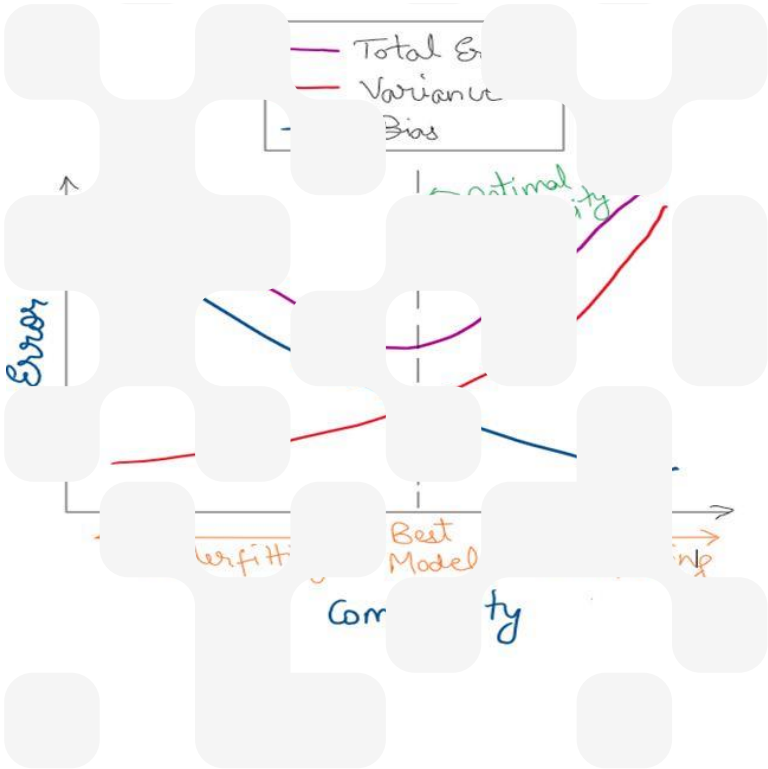


Agenda

I. GRADIENT BOOSTING

II. k-NEAREST NEIGHBORS (kNN)

III. SUPPORT VECTOR MACHINE (SVM)



Gradient boosting intuition

Consider a linear regression model $f = [x^{(i)}, \boldsymbol{\gamma}]$ parameterized with $\boldsymbol{\gamma}$ that maps each i-th input sample $x^{(i)}$ into the predictioin $F^{(i)}$ that should be close to the label $y^{(i)}$.

$$L(\boldsymbol{y}, \boldsymbol{\gamma}) = \frac{1}{2m}\sum_{i=1}^{m}\left(y^{(i)} - F^{(i)}\right)^2 \Rightarrow \min.$$

Consider a composition: $F_k = a_0 + a_1 + \cdots + a_k$.

Train $F_0 = a_0$ model (a simple decision tree) and check it's residual r_0.

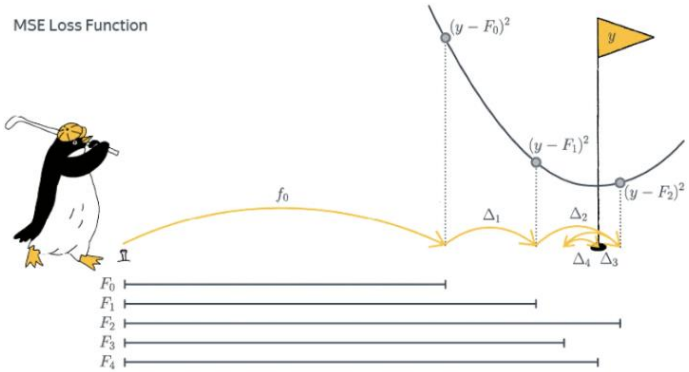
Train $F_1 = a_0 + a_1$ model to reduce the residual r_0 and check the residual r_1.

…

For example, $m = 1$:

$F_0 = a_0, r_0 = y - F_0$.

Suppose, $a_1 = -r_0$, then $F_1 = a_0 + a_1 = (y - r_0) - r_0 = y, r_1 = 0$.



Yandex Handbook on ML

Gradient boosting intuition

Consider a linear regression model $f = [x^{(i)}, \boldsymbol{\gamma}]$ parameterized with $\boldsymbol{\gamma}$ that maps each i-th input sample $x^{(i)}$ into the predictioin $F^{(i)}$ that should be close to the label $y^{(i)}$.

$$L(\boldsymbol{y}, \boldsymbol{\gamma}) = \frac{1}{2m}\sum_{i=1}^{m}\left(y^{(i)} - F^{(i)}\right)^2 \Rightarrow \min.$$

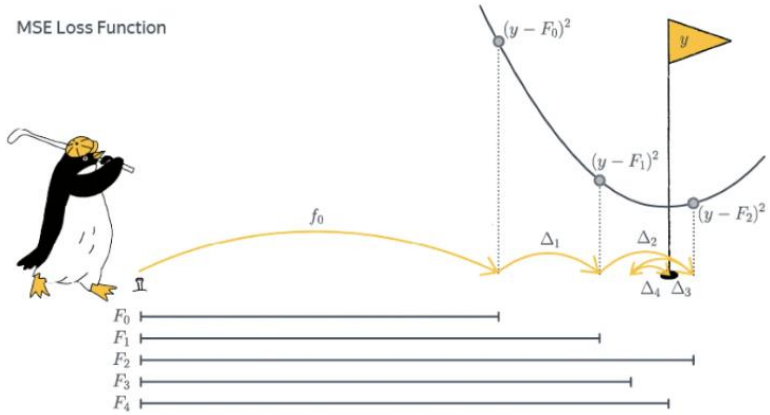
In a more general case, $m > 1$:

$$F_0 = \underset{\boldsymbol{\gamma}}{\operatorname{argmin}} L(\boldsymbol{y}, \boldsymbol{\gamma}),\ r_{i0} = y^{(i)} - F_0^{(i)}.$$

$$F_1 = F_0 + h_0 = F_0 + \underset{\boldsymbol{\gamma}}{\operatorname{argmin}} L(\boldsymbol{r}_0, \boldsymbol{\gamma}),\ r_{i1} = y^{(i)} - F_1^{(i)}.$$

...

$$F_k = F_{k-1} + h_k = F_{k-1} + \underset{\boldsymbol{\gamma}}{\operatorname{argmin}} L(\boldsymbol{r}_{k-1}, \boldsymbol{\gamma}),\ r_{ik} = y^{(i)} - F_k^{(i)}.$$



It can bee seen that the residual is the negative gradient of the loss : $-\left[\frac{\partial L}{\partial F_k}\right]_{F=F_k} = -\frac{1}{2m}\left[\frac{\partial}{\partial F_k}\left(\sum_{i=1}^{m}\left(y^{(i)} - F^{(i)}\right)^2\right)\right]_{F=F_k} = \boldsymbol{r}_k.$

Yandex Handbook on ML

Gradient boosting algorithm

Explanation:

1. **Initialize the Model**: Start with a simple model (often a constant value).
2. **Iteratively Add Models**: At each iteration, add a new model that attempts to correct the errors made by the current ensemble. **Fit a Weak Learner**: Fit a weak learner (e.g., a decision tree) to the pseudo-residuals. **Compute the Step Size**: Compute the optimal step size that minimizes the loss function. **Update the Model**: Update the model by adding the new weak learner with the computed step size.
3. **Final Model**: The final model is the combination of all the weak learners.

DeepSeek speaks

Step	Description
1. **Initialize the Model**	$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{m} L(y_i, \gamma)$$ This is typically the mean (for regression) or the log-odds (for binary classification).
2. **For each iteration** $k = 1$ **to** K	
a. **Compute Pseudo-Residuals**	$$r_{ik} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{k-1}(x)}$$ These are the negative gradients of the loss function with respect to the current model's predictions.
b. **Fit a Weak Learner**	Fit a weak learner (e.g., a decision tree) to the pseudo-residuals: $$h_k(x) = \arg\min_{h} \sum_{i=1}^{m} (r_{ik} - h(x_i))^2$$
c. **Compute the Step Size**	Compute the optimal step size α_k that minimizes the loss function: $$\alpha_k = \arg\min_{\alpha} \sum_{i=1}^{m} L(y_i, F_{k-1}(x_i) + \alpha h_k(x_i))$$
d. **Update the Model**	$$F_k(x) = F_{k-1}(x) + \alpha_k h_k(x)$$
3. **Final Model**	$$F(x) = F_K(x)$$

Why Gradient Boosting rules?

1. **High Predictive Accuracy.** GB is known for its ability to achieve high predictive accuracy. It does this by iteratively building an ensemble of weak learners (typically decision trees) and combining their predictions to create a strong learner. Each new tree is trained to correct the errors made by the previous trees, leading to a model that can capture complex patterns in the data.
2. **Flexibility.** GB can be applied to a wide range of loss functions, making it flexible for different types of problems. Any differentiable loss function can be used.
3. **Handles Various Data Types.** GB can handle numerical features, categorical features, missing values.
4. **Feature Importance.** GB provides a measure of feature importance, which helps in understanding which features are most influential in making predictions. This is useful for interpretability and feature selection.
5. **Regularization Techniques.** GB includes several regularization techniques to prevent overfitting. **Shrinkage**: A learning rate (or shrinkage factor) is applied to each tree's contribution reducing overfitting. **Subsampling**: Stochastic GB involves training each tree on a random subset of the data, which introduces randomness and reduces overfitting. **Tree Constraints**: Limiting the depth of the trees or the number of leaves can prevent the model from becoming too complex.
6. **Scalability**. Modern implementations of Gradient Boosting, such as XGBoost, LightGBM, and CatBoost, are highly optimized and can handle large datasets efficiently.
7. **Interpretability.** GB models can provide insights through feature importance and partial dependence plots.

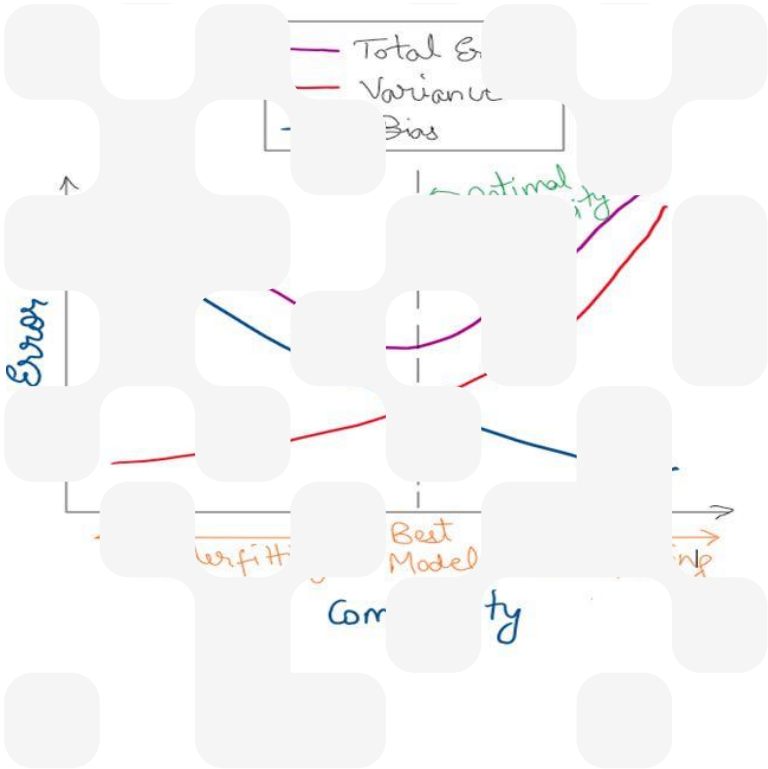
DeepSeek speaks

Agenda

I. GRADIENT BOOSTING

II. k-NEAREST NEIGHBORS (kNN)

III. SUPPORT VECTOR MACHINE (SVM)



k-Nearest Neighbors intuition

kNN is a simple and intuitive *metric* algorithm used for both *classification* and *regression* tasks. The core idea behind kNN is to classify a new data point based on the majority class of its k-nearest neighbors in the feature space.

- **Feature Space**: The space defined by the features (attributes) of the data points.
- **Distance Metric**: A measure of similarity between data points. Common distance metrics include:
  - **Euclidean Distance**: The straight-line distance between two points.
    $$d(p, q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$
  - **Manhattan Distance**: The sum of the absolute differences between the coordinates of the points.
    $$d(p, q) = \sum_{i=1}^{n} |p_i - q_i|$$
  - **Minkowski Distance**: A generalization of both Euclidean and Manhattan distances.
    $$d(p, q) = \left(\sum_{i=1}^{n} |p_i - q_i|^r\right)^{1/r}$$
- **k-Nearest Neighbors**: The k data points in the training set that are closest to the new data point.

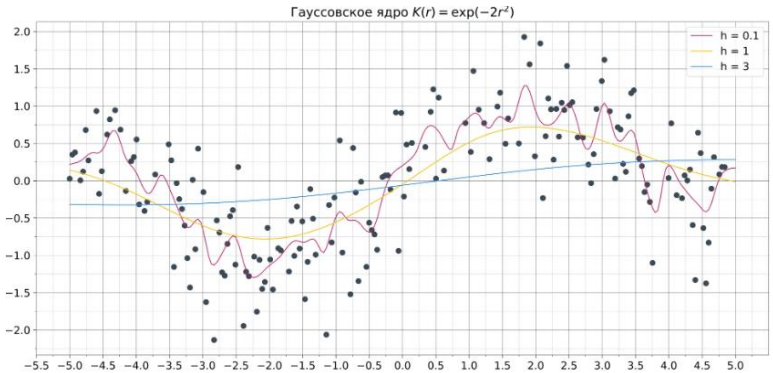
Yandex Handbook on ML

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load dataset
data = load_iris()
X = data.data
y = data.target

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X,
    y, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train a kNN classifier
k = 3  # Number of neighbors
model = KNeighborsClassifier(n_neighbors=k)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

DeepSeek writes a code

kNN algorithm

kNN is a simple and intuitive *metric* algorithm used for both *classification* and *regression* tasks. The core idea behind kNN is to classify a new data point based on the majority class of its k-nearest neighbors in the feature space.



Yandex Handbook on ML

1.Choose the Value of k: This is a hyperparameter that needs to be tuned.

2.Calculate Distances: For each new data point, calculate the distance to all training data points using a chosen distance metric.

3.Find k-Nearest Neighbors: Identify the k training data points with the smallest distances to the new data point.

4.Make a Prediction:

1. **Classification**: Assign the new data point to the class that is most common among its k-nearest neighbors (majority voting).
2. **Regression**: Predict the value of the new data point as the average of the values of its k-nearest neighbors.

DeepSeek speaks

kNN: advantages and disadvantages

- **Simple and Intuitive**: Easy to understand and implement.
- **No Training Phase**: The model does not require an explicit training phase; it uses the training data directly for prediction.
- **Non-Parametric**: Does not make any assumptions about the underlying data distribution.

- **Computationally Expensive**: Requires calculating distances to all training data points, which can be slow for large datasets.
- **Memory Intensive**: Needs to store the entire training dataset.
- **Sensitive to Irrelevant Features**: Performance can degrade if the feature space contains irrelevant or noisy features.
- **Choice of k**: The performance of kNN depends on the choice of k. A small k can lead to overfitting, while a large k can lead to underfitting.

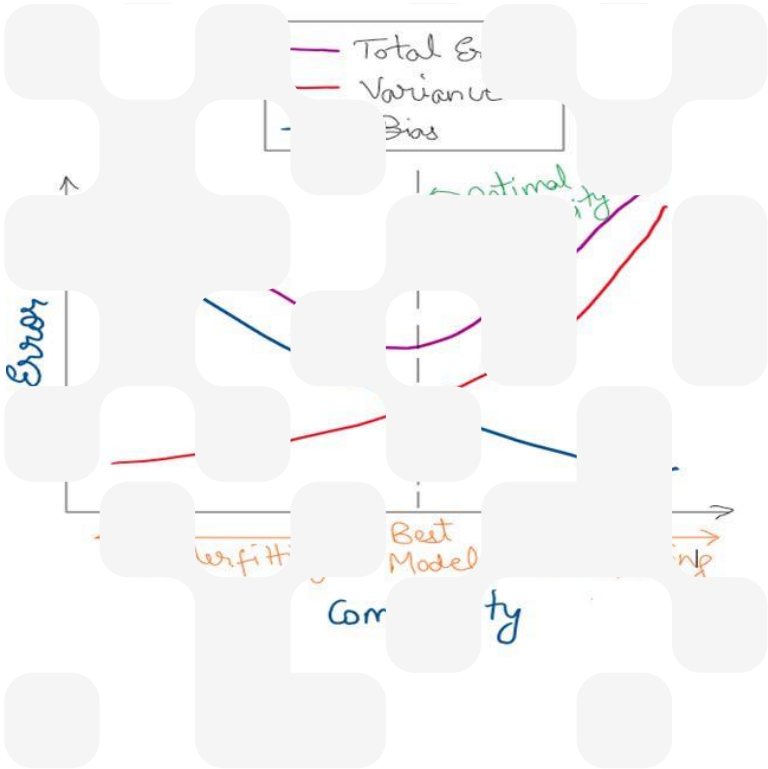
DeepSeek speaks

Agenda

I. GRADIENT BOOSTING

II. k-NEAREST NEIGHBORS (kNN)

III. SUPPORT VECTOR MACHINE (SVM)



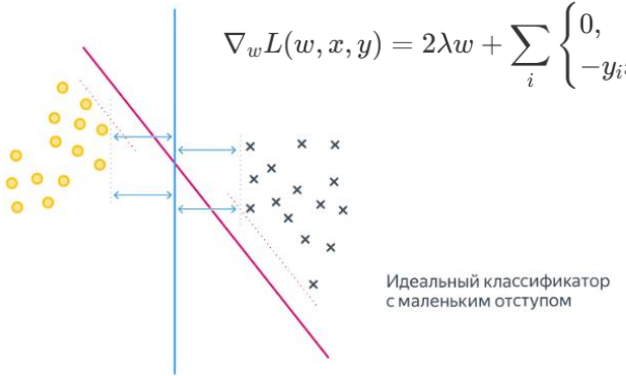
Support Vector Machine intuition

SVM is a linear method which aims to find the hyperplane that maximizes the *margin* between the classes. The margin is the distance between the hyperplane and the nearest data points from each class, known as support vectors. The goal is to find the hyperplane that provides the best generalization to unseen data.

$$F(M) = \max(0, 1 - M)$$

$$L(w, x, y) = \lambda \|w\|_2^2 + \sum_i \max(0, 1 - y_i \langle w, x_i \rangle)$$

$$\nabla_w L(w, x, y) = 2\lambda w + \sum_i \begin{cases} 0, & 1 - y_i \langle w, x_i \rangle \leq 0 \\ -y_i x_i, & 1 - y_i \langle w, x_i \rangle > 0 \end{cases}$$



Yandex Handbook on ML

```python
from sklearn import svm
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load dataset
data = load_iris()
X = data.data
y = data.target

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X,
    y, test_size=0.2, random_state=42)

# Train an SVM model
model = svm.SVC(kernel='linear', C=1.0)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

DeepSeek writes a code

SVM: advantages and disadvantages

- Effective in High-Dimensional Spaces: Works well even when the number of dimensions is greater than the number of samples.
- Memory Efficient: Uses only a subset of the training points (support vectors) in the decision function.
- Versatile: Can be used for both classification and regression tasks.

- Sensitive to Noise: A few misclassified points can significantly affect the hyperplane.
- Computationally Expensive: Training can be slow for large datasets.
- Requires Tuning: Hyperparameters like the kernel type, kernel parameters, and regularization parameter CC need to be tuned.

DeepSeek speaks

Thank you for your attention!

a.kornaev@innopolis.ru, @avkornaev