Final Project - Theory Recap

CLASSIFIERS

Bellini Emanuele, Karakoseian Vincent, Meanti Luca, Nevander Eddie, Rossoni Guido

Contents

1. K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN)

Task: classify the class of a data point based on the classes of its nearest neighbors in the feature space. It aims to assign a class label to a data point based on the majority class among its K nearest neighbor.

Algorithm:

1. **Data Preparation**: select a dataset with features and corresponding labels. Features are the attributes or properties of the data points, while labels are the classes or values to be predicted.
2. **Choose K**: Decide the number of neighbors (K) to consider when making predictions. This is typically determined through experimentation or cross-validation.
3. **Calculate Distances**: For a given data point that must be classified or predicted, calculate the distance between that point and every other point in the dataset. The distance metric commonly used is *Euclidean distance*, alternatively *Hamming* or *Mahalanobis*, but other metrics like *Manhattan* distance or cosine similarity can also be used.
4. **Find Nearest Neighbors**: Select the K data points with the shortest distances to the point to be classified.
5. **Majority Vote**: Determine the majority class among the K nearest neighbors and assign that class to the data point.
6. **Evaluate Model Performance**: After making predictions on a test set or using cross-validation, evaluate the performance of the KNN algorithm using appropriate evaluation metrics like accuracy, precision, recall or F1-score.

It's a non-parametric and lazy learning algorithm, meaning it doesn't make assumptions about the underlying data distribution and doesn't require a training phase until prediction time. Instead, it simply memorizes the training data and performs computations at the time of prediction.

One important thing to note about KNN is that it's sensitive to the choice of K and the distance metric used. A small value of K can lead to noise affecting the predictions, while a large value of K can cause the algorithm to overlook local patterns.

Additionally, the choice of distance metric depends on the nature of the data and the problem at hand.

2. Logistic Regression (LR)

Logistic Regression (LR)

Task: model the probability that an instance belongs to a particular class in a binary classification problem. It is typically used when the dependent variable (or target variable) is categorical with two possible outcomes represented as 0 and 1.

Algorithm:

1. **Data Preparation**: Dataset consisting of input features (independent variables) and corresponding binary class labels (dependent variable). This dataset is typically split into training and testing sets for model training and evaluation, respectively.
2. **Model Representation**: The relationship between the input features X and the binary class label y is modeled using a logistic function, also known as the "sigmoid" function:
$$g(z) = \frac{1}{1+e^{-z}}$$
where $z = \theta_0 + \theta_1 x_1 + ... + \theta_n x_n$ is a linear combination of the input features and model parameters.
3. **Training the Model**: The parameters $\underline{\theta}$ are learned from the training data using optimization techniques such as Gradient Descent or its variants. The objective is to find the optimal values of $\underline{\theta}$ that maximizes the total number of positive outcomes.
4. **Predicting Probabilities**: The logistic function is used to predict the probability that an instance belongs to the positive class:
$$P(y=1|x,\underline{\theta}) = \frac{1}{1+e^{-\underline{\theta}^T x}}$$
5. **Thresholding**: To obtain binary predictions, a threshold $(\bar{z})$ is applied to the predicted probabilities. $\begin{cases} \hat{y} = 1 & if \quad p > \bar{z} \\ \hat{y} = 0 & if \quad p < \bar{z} \end{cases}$
6. **Regularization**: Regularization techniques such as Lasso or Ridge regularization can be applied to penalize large parameter values.
7. **Evaluation**: The performance of the logistic regression model is evaluated using metrics such as accuracy, precision and ROC-AUC on a separate validation or test dataset.

LR is primarily used for binary classification tasks, but it also provides the probability that an instance belongs to a particular class.

The coefficients of the model indicate the strength of the relationship between the input features and the log-odds of the positive class. This interpretability is particularly valuable in fields where understanding the factors influencing the outcome is essential, such as healthcare, social sciences, and finance.

3. Support Vector Machine (SVM)

Support Vector Machine (SVM)

Task: find the optimal hyperplane that best separates the data into different classes while maximizing the margin between them. SVM aims to create a decision boundary that maximizes the separation between data points of different classes.

Algorithm for binary classification:

1. **Feature Scaling**: Scale the input features through *standardization* (subtracting the mean and dividing by the standard deviation) or *normalization* (scaling to a range between 0 and 1).
2. **Kernel Matrix Computation**: Compute the kernel matrix K based on the chosen kernel function and the input feature vectors $x^{(i)}$. If using a linear kernel, this step is skipped as the kernel matrix is simply the inner product of the feature vectors.
3. **Quadratic Programming (QP) Formulation**: Formulate the SVM optimization problem as a quadratic programming (QP) problem:
$$\min \frac{1}{2}\underline{\theta}^T\underline{\theta} + C\sum_{i=1}^{m}\xi_i$$
subject to
$$y_i(\underline{\theta}^T\phi(x_i) + b) \geq 1 - \xi_i, \text{ for i=1, ..., m}$$
$$\xi_i \geq 0, \text{ for i=1, ..., m}$$
where C is the regularization parameter that controls the trade-off between maximizing the margin and minimizing the classification error, x_i represents the input features of the i-th instance and y_i represents its corresponding class label (0 or 1), ξ_i are stack variables, $\underline{\theta}$ are the model parameters, $\phi(x_i)$ represents the feature mapping function and b is the bias term.
4. **Solve QP Problem**: Use optimization techniques (e.g., Sequential Minimal Optimization) to solve the QP problem and find the optimal values of $\underline{\theta}$ and b.
5. **Extract Support Vectors**: Identify the support vectors from the training dataset. These are the data points that lie on or within the margin boundary.
6. **Compute Decision Boundary**: Use the learned parameters $\underline{\theta}$ and b to compute the decision boundary and margin of the SVM model.
7. **Model Evaluation**: Evaluate the performance of the trained SVM model using appropriate evaluation metrics such as accuracy, precision and ROC-AUC on a separate validation or test dataset.
8. **Output**: Return the learned parameters $\underline{\theta}$ of the SVM model.

The core idea behind SVM is to find the hyperplane that best separates the data into different classes while maximizing the margin between them. The margin is the distance between the hyperplane and the nearest data points from each class, also known as support vectors.

This technique can efficiently handle non-linear classification problems by implicitly mapping the input features (that can be both categorical and continuous) into a higher-dimensional space using a kernel function. This allows SVM to find a linear decision boundary in the transformed feature space, even when the original data is not linearly separable.

SVM supports different kernel functions such as linear, polynomial, radial basis function (RBF), and sigmoid. The choice of kernel function depends on the nature of the data and the problem at hand. RBF kernel is commonly used due to its flexibility in capturing complex relationships in the data.

Here are some key aspects of SVM's purpose in classification problems:

- **Maximizing Margin**: SVM seeks to find the hyperplane that maximizes the margin. By maximizing the margin, SVM aims to achieve better generalization performance and robustness to variations in the data.

- **Handling Non-Linear Separable Data**: SVM can handle both linearly and non-linearly separable data by implicitly mapping the input features into a higher-dimensional space using a kernel function. This allows SVM to find a linear decision boundary in the transformed feature space, even when the original data is not linearly separable.

- **Robustness to Outliers**: SVM is relatively robust to outliers in the data, as it primarily focuses on the support vectors, which are the data points closest to the decision boundary. Outliers that are not support vectors have little impact on the placement of the decision boundary.

- **Effective in High-Dimensional Spaces**: SVM performs well in high-dimensional spaces, making it suitable for classification problems with a large number of features. This makes SVM particularly useful in text classification, image classification, and bioinformatics, where data often have high dimensionality.

- **Interpretability**: While SVMs are not as interpretable as some other algorithms like Decision Trees, they still provide insights into the decision boundary and the importance of different features through the support vectors and coefficients of the model.

4. Isolation Forest

Isolation Forest

Task: unsupervised learning algorithm that identifies anomalies (or outliers) in datasets by isolating them in a tree-based structure.

Algorithm:

1. **Tree Construction (Isolation Tree)**:
   - Randomly select a subset D' of the dataset D containing a fixed number of data points.
   - Randomly select a feature q and a split value p for the selected feature such that p lies between the minimum and maximum values of the selected feature.
   - Split the data points into two disjoint subsets: one subset containing data points with values less than or equal to p, and the other subset containing data points with values greater than p.
   - Recursively repeat the above steps until one of the following conditions is met: the maximum depth of the tree is reached, all data points in the current subset are the same (i.e., no further splits are possible) or the current subset contains only one data point.
2. **Forest Construction**: Repeat the above step *n_estimators* times to construct a forest of isolation trees. *n_estimators* represents the number of trees in the forest.
3. **Anomaly Score Calculation**: For each data point x_i, calculate its anomaly score $S(x_i)$ based on the average path length in the isolation trees:
   $$S(x_i) = 2^{-\frac{\mathbb{E}[h(x_i)]}{c(n)}}$$
   where
   - $h(x_i)$ is the average path length of x_i in all trees.
   - $\mathbb{E}[h(x_i)]$ is the average path length of x_i in the forest.
   - $c(n)$ is the average path length of an unsuccessful search in a binary tree, which is approximately $2log(n-1) - \frac{2(n-1)}{n}$ for n data points.
4. **Anomaly Identification**: Set a threshold T to classify data points as anomalies or normal data points based on their anomaly scores. Data points with anomaly scores higher than T are considered anomalies.

Isolation Forest is scalable to large datasets due to its tree-based structure and the ability to construct trees in parallel. Anomalies are identified more quickly than normal data points, as they typically require fewer splits to isolate; moreover, it performs well even in high-dimensional data spaces, making it suitable for a wide range of applications.

5. Multi-layer Perceptron (MLP)

Multi-layer Perceptron (MLP)

Task: type of artificial neural network that consists of multiple layers of interconnected neurons (or units). It's a versatile and powerful model capable of learning complex patterns and relationships in data.

Algorithm:

1. **Architecture**: MLP consists of an input layer, one or more hidden layers, and an output layer. Each layer (except the input layer) contains multiple neurons, also known as units or nodes. Neurons in adjacent layers are fully connected, meaning each neuron in one layer is connected to every neuron in the next layer.
2. **Forward Propagation**: During forward propagation, input data is fed into the network, and computations are performed layer by layer to generate predictions. Each neuron computes a weighted sum of its inputs, applies an activation function, and passes the result to the next layer.
3. **Activation Functions**: Activation functions introduce non-linearity into the network, allowing MLP to learn complex relationships in data. Common activation functions include the sigmoid function, the hyperbolic tangent and the Rectified Linear Unit (ReLU).
4. **Loss Function**: The choice of loss function depends on the task; for classification tasks, the cross-entropy loss function is commonly used.
5. **Backpropagation**: Backpropagation is used to update the weights of the network to minimize the loss function. Gradients of the loss function with respect to the weights are computed using the chain rule of calculus. The weights are updated using an optimization algorithm such as stochastic gradient descent (SGD), Adam, or RMSprop.
6. **Training**: MLP is trained using labeled data through an iterative process of forward propagation and backpropagation. The training process aims to minimize the loss function on the training data by adjusting the weights of the network.
7. **Regularization**: Regularization techniques such as Lasso, Ridge regularization or dropout can be applied to prevent overfitting and improve generalization performance.
8. **Hyperparameter Tuning**: Hyperparameters such as the number of hidden layers, the number of neurons in each layer, learning rate, activation functions, and regularization strength need to be tuned to optimize the performance of the MLP.
9. **Evaluation**: The performance of the trained MLP model is evaluated using appropriate metrics such as accuracy and precision on a separate validation or test dataset.
10. **Prediction**: The MLP model can be used to make predictions on new, unseen data by performing forward propagation with the learned weights.

Here are some key characteristics about Multi-layer Perceptron:

- **Versatility**: MLP can be applied to a wide range of tasks, including classification, regression, and even unsupervised learning tasks such as clustering and dimensionality reduction.
- **Non-linear Modeling**: MLP can learn complex non-linear relationships in data, making it suitable for tasks with intricate patterns.
- **Scalability**: MLP can scale to large datasets and high-dimensional feature spaces, although training may become computationally intensive for very large networks.
- **Interpretability**: While MLP can achieve high performance, the inner workings of the model may be less interpretable compared to simpler models like logistic regression or decision trees.

Overall, the Multi-Layer Perceptron is a versatile and powerful neural network model that has been successfully applied to a wide range of machine learning tasks. Its ability to learn complex patterns and relationships makes it a valuable tool in the field of artificial intelligence and data science.

6. Gradient Boosting

Gradient Boosting

Task: Gradient Boosting is a machine learning technique that builds a strong predictive model by combining the predictions of multiple weak models, typically decision trees.

Algorithm:

1. **Initialization**: initializing the model with a simple prediction of a constant value for classification.
2. **Iterative Training**: The model is trained iteratively, where each decision tree is fitted to the residuals (or gradients) of the previous model's predictions. The residuals are typically the negative gradients of the loss function with respect to the predicted probabilities.
3. **Building Trees**: Each tree is typically shallow (weak learner) to prevent overfitting. Trees are added sequentially until a predefined number of trees (or until a specified stopping criterion) is reached. Each new tree improves the performance of the ensemble model by reducing the residuals of the previous model.
4. **Gradient Descent Optimization**: The model parameters are optimized using Gradient Descent (or a similar optimization algorithm) to minimize the loss function that measures the discrepancy between the actual target values and the predictions of the ensemble model. This can be the Binary Cross-Entropy Loss (used for binary classification tasks, where the target variable has two classes 0 and 1) or the Log Loss (Logistic Loss, similar to binary cross-entropy loss, often used in logistic regression and binary classification tasks).
5. **Prediction**: The final ensemble model aggregates the predictions of all the individual trees by summing them together and the final prediction is obtained by applying a softmax function to the sum of the predictions.
6. **Regularization**: Gradient Boosting models can be prone to overfitting, especially if the individual trees are too complex or if too many trees are added. Regularization techniques such as shrinkage (learning rate) and tree pruning are often used to prevent overfitting and improve generalization performance.

Gradient Boosting combines the predictions of multiple weak models (typically decision trees) to create a strong predictive model. The models are trained sequentially, with each new model focusing on the residuals (or gradients) of the previous model's predictions.

Gradient Boosting optimizes the model parameters using gradient descent or a similar optimization algorithm to minimize a loss function. It is robust to noisy data and can handle a variety of data types and distributions. While individual trees in Gradient Boosting may be interpretable, the overall ensemble model may be less interpretable due to the complexity of combining multiple models.

7. Extreme Gradient Boost (XGBoost)

Extreme Gradient Boost (XGBoost)

Task: advanced implementation of the Gradient Boosting algorithm that is known for its efficiency, scalability, and high performance. It leverages advanced algorithms and data structures to improve training speed and performance, optimizing the original Gradient Boosting algorithm by introducing additional features such as regularization, parallel processing, and tree pruning.

Algorithm:

1. **Initialization**: Initializing the model with a simple prediction of a constant value for classification.
2. **Compute Initial Residuals**: Compute the initial residual by subtracting the initial prediction from the true target value.
3. **Iterative Training**: For $t = 1$ to T (number of boosting rounds):
   - Build a Tree: Construct a decision tree to predict the residuals from the previous iteration. Use a specialized algorithm (e.g., histogram-based approximation) for efficient tree construction and regularize the tree using techniques such as depth control, minimum child weight and maximum number of nodes.
   - Update Predictions: Update the predictions by adding the output of the new tree to the previous predictions. The learning rate η controls the contribution of each tree to the final prediction.
   - Compute Residuals: Compute the residuals for each sample by subtracting the updated predictions from the true target values.
4. **Regularization**: Apply regularization techniques such as Lasso and Ridge regularization to prevent overfitting. Regularization penalties are added to the objective function during training to penalize complex models.
5. **Objective Function Optimization**: Define an objective function (made of two parts, a loss term and a regularization term) that measures the discrepancy between the predictions and the true target values. Optimize it using gradient descent or another optimization algorithm. Stop training if the performance does not improve for a certain number of iterations (early stopping) to prevent overfitting.
6. **Prediction**: Aggregate the predictions of all the trees by summing them together and apply a softmax function to the sum of the predictions to obtain class probabilities.

XGBoost is highly optimized for speed and memory usage, making it suitable for large-scale machine learning tasks. It consistently achieves high predictive performance and is competitive with state-of-the-art algorithms. In conclusion, it supports various objective functions, evaluation metrics, and advanced features to handle diverse machine learning tasks.

8. Decision Trees and AdaBoost

Decision Trees and AdaBoost

Task: powerful and widely used ensemble learning algorithm that combines multiple weak learners in a sequential and adaptive manner, creating a strong classifier that often outperforms individual base learners. Its simplicity, effectiveness, and versatility make AdaBoost a popular choice in both research and practical applications. AdaBoost combined with Decision Trees is robust to overfitting and noisy data, as it focuses more on samples that are difficult to classify.

Algorithm:

1. **Initialization**: Initialize the weights of training samples uniformly, so each sample has an equal weight initially, choosing a decision tree as the base classifier.
2. **Iterative Training**: For $t = 1$ to T (number of boosting rounds):
   - Train the Decision Tree: train a decision tree on the training data, where the samples are weighted according to their current weights, minimizing the weighted classification error, focusing more on the samples that were misclassified in previous iterations.
   - Compute Error: compute the weighted error of the decision tree on the training data as the sum of weights of misclassified samples divided by the total weight of all samples.
   - Compute Learner Weight: compute the weight of the decision tree in the final ensemble based on the error rate, with lower error rates resulting in higher weights.
   - Update Sample Weights: update the weights by increasing the weights of misclassified samples and decreasing the weights of correctly classified samples.
   - Normalize Weights: Normalize the sample weights so that they sum to one, ensuring that they represent a valid probability distribution.
3. **Final Prediction**: Combine the predictions of all decision trees by weighted majority voting. The final prediction is obtained by summing the weighted predictions of all decision trees and applying a threshold to determine the class label.

There are some considerable advantages of AdaBoost with Decision Trees:

- **Ensemble Learning**: AdaBoost with decision trees combines multiple weak learners (decision trees) to create a strong learner.
- **Adaptive Learning**: AdaBoost adapts to the training data by adjusting the weights of samples based on their classification errors.

- **Robustness**: AdaBoost with decision trees is robust to overfitting and noisy data, as it focuses more on samples that are difficult to classify.
- **Interpretability**: Decision trees used as base classifiers are easy to interpret and understand, making it easier to explain the final model to stakeholders.

In conclusion, decision trees used in AdaBoost are typically shallow to prevent overfitting and improve generalization performance. Regularization techniques such as limiting the maximum depth of trees or pruning can be applied to prevent overfitting; tuning hyperparameters of the decision tree, such as the maximum depth, minimum samples per leaf, and splitting criteria, can impact the performance of AdaBoost with decision trees.

By combining Decision Trees with AdaBoost, it is possible to create a powerful ensemble model that leverages the strengths of both algorithms, resulting in improved predictive performance and robustness.

9. Naive Bayes classifier

Naive Bayes classifier

Task: probabilistic machine learning algorithm based on Bayes' theorem. It is commonly used for classification tasks, especially in text classification and spam filtering. It often performs surprisingly well in practice and serves as a baseline model for many classification tasks.

Algorithm:

1. **Naive Assumption**: The "naive" assumption in Naive Bayes is that the features are conditionally independent given the class label. This means that the presence or absence of a particular feature is assumed to be independent of the presence or absence of any other feature, given the class label. Mathematically, this can be written as:

$$P(x_1, x_2, ..., x_n|y) = P(x_1|y) \times P(x_2|y) \times ... \times P(x_n|y)$$

2. **Model Training**: Calculate the prior probabilities $P(y)$ for each class by counting the frequency of each class in the training data.
Estimate the likelihood $P(x|y)$ for each feature given each class by counting the frequency of each feature value in the training data for each class.
3. **Model Prediction**: Given a new instance with features x, calculate the posterior probability $P(y|x)$ for each class using Bayes' theorem. The class with the highest posterior probability is predicted as the class label for the new instance:

$$\hat{y} = \arg\max_{y} \; P(y|x)$$

Types of Naive Bayes Classifiers:

- Gaussian Naive Bayes: Assumes that continuous features follow a Gaussian (normal) distribution.
- Multinomial Naive Bayes: Used for discrete features, typically in text classification tasks where features represent word counts or frequencies.
- Bernoulli Naive Bayes: Similar to multinomial Naive Bayes but assumes that features are binary variables.

Advantages of Naive Bayes Classifier:

- `Simplicity`: Naive Bayes is simple to implement and easy to understand.
- `Efficiency`: It is computationally efficient and scales well to large datasets and high-dimensional feature spaces.
- `Robustness`: Naive Bayes can handle noisy data and irrelevant features well.

- `Interpretability`: The probabilistic nature of Naive Bayes allows for easy interpretation of results.

Limitations of Naive Bayes Classifier:

- `Naive Assumption`: The assumption of feature independence may not hold true in real-world datasets, leading to suboptimal performance.
- `Handling Continuous Features`: Gaussian Naive Bayes assumes continuous features follow a normal distribution, which may not always be appropriate.
- `Data Scarcity`: Naive Bayes may perform poorly with small datasets or when certain feature-label combinations have no occurrences in the training data.