BJ to Lab-2-3:

**Orange system**

Download and install:

<https://orangedatamining.com/>

**Windows**

**Standalone installer (default)**

<https://orangedatamining.com/>

Here is a short descripton

Downloding:

<https://orangedatamining.com/download/>

use this:

[Orange3-3.36.2-Miniconda-x86\_64.exe (64 bit)](https://download.biolab.si/download/files/Orange3-3.36.2-Miniconda-x86_64.exe)

Can be used without administrative privileges.

See (!) Getting Started with Orange

<https://www.youtube.com/watch?v=HXjnDIgGDuI&list=PLmNPvQr9Tf-ZSDLwOzxpvY-HrE0yv-8Fy&index=2>

Those are the video-lectures (how to start,…. To work with different methods)

The same as above: In Orange 🡪Help🡪 video tutorials (!) (or + Documentation)

***Some asked questions:***

***Bar Plot****: Def*.: A bar plot is a graphical representation of data where individual bars or columns represent different categories or groups. The length (or height) of each bar corresponds to the quantity or frequency of the data it represents.

*Use Case*: Bar plots are often used to compare the values of different categories or show the distribution of a categorical variable. They are suitable for visualizing discrete data.

***Violin Plot (recommendation: It is not easy to understand the joint distribution densities along the x axes and y categories. I’d recommend not using this widget)***:

*Def*.: A violin plot is a method of plotting numeric data and can be considered as a combination of a box plot and a kernel density plot. It displays the distribution of the data along with its probability density. The thicker areas of the plot represent where the data points are more concentrated.

***A kernel density plot***, often simply referred to as a kernel density estimate (KDE) plot, is a non-parametric way to estimate the probability density function of a random variable. It is a smoothed representation of a histogram and provides a continuous curve that approximates the underlying probability distribution of the data.

***Kernel density plots:***

*Kernel Function*: The kernel function is a smooth, symmetric, and non-negative function. It is centered at each data point and contributes to the overall density estimation. Common kernel functions include Gaussian (normal) and others.

*Bandwidth*: The bandwidth controls the width of the kernel and, consequently, the smoothness of the resulting curve. A smaller bandwidth leads to a more sensitive, detailed plot, while a larger bandwidth results in a smoother, more generalized plot.

*Procedure*: For each data point, a kernel function is centered at that point, and the contributions from all kernels are summed to create a smoothed estimate of the probability density.

*Visualization*: The kernel density plot is often visualized as a continuous curve. It provides insights into the underlying distribution of the data, indicating where the data is more concentrated (peaks) and where it is less dense (troughs).

*Use Case*: Kernel density plots are particularly useful for visualizing the shape and characteristics of a distribution when the underlying probability distribution is not known. They provide a smooth and informative representation of the data's density.

*In summary*, a kernel density plot is a valuable tool for understanding the distribution of data, especially when you want a smooth representation that goes beyond the granularity of a histogram.

**Normalization of the input matrix:**

1. If we do not normalize the values of the criteria/features, located in the columns of the data matrix (rows are samples), can calculating distances between samples/rows be considered meaningless if the criteria have completely different meanings (for example, weight, height, salary...).

If the values of the criteria have completely different meanings and scales, calculating distances between samples without normalization can lead to distorted results. Distances such as Euclidean distance or cosine distance often depend on the scale of the features. Therefore, data normalization (standardization) can be an important preprocessing step.

Normalizing data allows bringing the feature values to a common scale, making distances more comparable and improving the performance of many machine learning methods. This is especially crucial for distance-based methods like k-Nearest Neighbors (k-NN) or clustering algorithms.

Orange provides tools for data normalization, and using them can significantly enhance the results of your data analysis.

1. **In Orange, five normalization/standardization methods (of columns) are offered. How much can the final decision (conclusions) depend on the normalization method?**

The choice of normalization method can have an impact on the performance of certain machine learning algorithms. Different methods handle data transformation differently, and the nature of your data and the algorithm you plan to use may influence the effectiveness of one method over another.

* + **Normalization** (several variants: Min-Max Scaling) to [0,1]: Scales the data to a specific range, which can be useful for algorithms sensitive to the scale of features.
  + **Z-Score (Standard Score):** Standardizes data to have a mean of 0 and a standard deviation of 1. This is effective when features have different units or scales.
  + **Log Transformation:** Useful for handling right-skewed distributions and making the data more symmetrical.
  + **Box-Cox Transformation:** Generalizes the log transformation and can handle data with different types of skewness.
  + **Any Transformation:** Allows applying a custom transformation function.

The choice of method depends on the characteristics of your data and the specific requirements of the algorithm you are using. It's often a good practice to try different normalization methods and assess their impact on model performance during the model evaluation process. The output results can depend on the normalization method.

**Hierarchical Clustering widget:**

**Max Depth and Top N in Hierarchical Clustering:**

Hierarchical clustering algorithms build a hierarchy of clusters by iteratively merging or splitting data points. The options you mentioned control the structure and granularity of the resulting hierarchy.

**1. Max Depth:**

* **Function:** Sets the maximum **depth** (number of merging/splitting steps) allowed in the hierarchy.
* **Impact:**
  + A **higher Max Depth** results in a **deeper hierarchy** with more granular clusters. Each cluster may contain fewer data points but might be more homogeneous (similar within the cluster).
  + A **lower Max Depth** leads to a **shallower hierarchy** with fewer, more general clusters. Each cluster might contain more data points but might be less homogeneous.
* **Choosing Max Depth:**
  + There's no single "best" value. Experiment with different values and evaluate the resulting clusters based on your specific needs and data.
  + Consider factors like the desired level of detail and interpretability of the clusters.

**2. Top N:**

* **Function:** Specifies the maximum number of clusters (**N**) to display at the **bottom level** of the hierarchy.
* **Impact:**
  + Shows only the **N most specific clusters** in the final hierarchy.
  + Helps to **focus** on the most differentiated clusters.
  + Useful when dealing with very large datasets to avoid overwhelming visualizations.
* **Choosing Top N:**
  + Consider the desired level of granularity and the number of meaningful clusters present in your data.
  + Analyze visualizations (e.g., dendrogram) to see how distinct the clusters become at different levels.

**Ranking in Hierarchical Clustering:**

Hierarchical clustering doesn't explicitly rank **classes** in the traditional sense (like ranking students by grades). Instead, it builds a **hierarchy** that shows how clusters are formed by merging or splitting data points based on their similarity.

* **Interpretation:**
  + Clusters at the **bottom level** (highest depth) are the **most specific** and represent the most differentiated groups in the data.
  + As you move **up** the hierarchy (lower depth), clusters **merge**, becoming more general and encompassing more data points.

**Summary:**

* **Max Depth:** Controls the overall depth and granularity of the hierarchy.
* **Top N:** Focuses on the most specific clusters at the bottom level.
* Hierarchical clustering doesn't rank classes directly, but the structure of the hierarchy reveals differentiated groups based on their similarity.

In scatter plots, **jittering** is a technique used to address overplotting, which occurs when multiple data points overlap at the same location in the plot. This overlapping can obscure patterns and make it difficult to interpret the data visually.

**Here's how jittering works:**

* **Small random values:** A small amount of random noise is added to the x and/or y coordinates of each data point. This slight shift spreads the points out slightly, reducing the chance of them completely overlapping.
* **Visualization improvement:** By adding jitter, data points become more distinguishable, allowing you to see the overall distribution and potential clusters or trends in the data more clearly.

**Benefits of Jittering:**

* **Reduced overplotting:** Makes it easier to identify individual data points, even when they have similar values.
* **Improved visualization:** Helps reveal patterns and trends that might be hidden by overlapping points.
* **Clarity:** Enhances the overall clarity and interpretability of the scatter plot.

**Overall, jittering is a simple yet effective way to enhance the visualization of scatter plots, especially when dealing with dense data and potential overplotting issues**

**The simplest classifier problem involves building a classifier to predict the class label of instances based on a set of features**. Let's break down the components:

1. **Problem Description:**
   * **Goal:** The primary goal is to accurately classify instances into predefined classes or categories based on their features.
   * **Example Scenario:** Consider a dataset of emails, where the goal is to classify each email as either "Spam" or "Not Spam" based on features such as the presence of certain keywords, email sender, and other relevant attributes.
2. **Input Information:**
   * **Features (Independent Variables):** These are the attributes or characteristics of the instances that the classifier will use for prediction. For instance, in the email example, features could include word frequencies, sender information, etc.
   * **Class Label (Dependent Variable):** This is the variable we want to predict. In a binary classification problem, it typically represents the categories, such as "Spam" or "Not Spam."
3. **Output Information:**
   * **Predicted Class Labels:** The classifier's output is the predicted class label for each instance in the dataset. For example, after training, the classifier should be able to predict whether a new email is likely to be spam or not.

Now, let's discuss the steps to solve the simplest classifier problem using Orange:

1. **Data Loading:**
   * Load your dataset into Orange. This can be done using various file formats (e.g., CSV, Excel) or by using Orange's built-in datasets.
2. **Data Exploration:**
   * Explore the dataset to understand its structure, the types of features, and the distribution of class labels.
3. **Data Preprocessing:**
   * Handle missing values, if any.
   * Encode categorical variables, if present.
   * Normalize or standardize numeric features if needed.
4. **Data Splitting:**
   * Split the dataset into training and testing sets. The training set is used to train the classifier, and the testing set is used to evaluate its performance.
5. **Classifier Selection:**
   * Choose a classifier algorithm suitable for your problem. Orange offers a variety of classifiers, including decision trees, support vector machines, **k-nearest neighbors**, etc.
6. **Model Training:**
   * Train the selected classifier using the training set.
7. **Model Evaluation:**
   * Evaluate the performance of the trained model on the testing set. Common metrics include accuracy, precision, recall, and F1-score.
8. **Prediction:**
   * Use the trained classifier to make predictions on new, unseen instances.
9. **Model Interpretation:**
   * Understand the learned model, especially if it's a transparent model like a decision tree.
10. **Fine-Tuning (Optional):**
    * Depending on the results, you may fine-tune the model or try different classifiers to improve performance.
11. **Deployment (Optional):**
    * If the model meets the desired performance, it can be deployed for making predictions on new, real-world data.

This process provides a general guideline, and the specific steps may vary depending on the characteristics of your dataset and problem

Scatter Plot visualizes points based on their coordinates on the axes, *and if the data on the axes is categorical,* then Orange is likely using what's called ***"discrete jitter***" for better plot readability.

**Adding kNN:**

* **Preprocessed Data:** This provides the cleaned and prepared data for analysis.
* **Test and Score:** This widget splits the data into training and testing sets for kNN model evaluation.
  + Folds = 2 means the data will be divided into two folds (subsets).
* **kNN Widget (Left):** This widget trains the kNN model using the training set from Test and Score.
* **Confusion Matrix (Right):** This widget visually shows the performance of the kNN model on the testing set.

**2. Cross-Validation:**

* **Number of Folds = 2:** In this case, your data will be split into two roughly equal-sized folds.
  + One fold (training set) is used to train the kNN model.
  + The other fold (testing set) is used to evaluate the model's performance.
  + This process is repeated by swapping the roles of the folds, resulting in training on both sets and evaluating on both sets (cross-validation).
* **Stratified:** This option is relevant for classification tasks where you want to ensure the class proportions are preserved in each fold. Here, the focus is on evaluating how well the kNN model performs on unseen data (testing set).

**3. Random Sampling:**

* **Replicate Train/Test:** This option controls how many times the random sampling process is repeated. Setting it to a higher value (e.g., 10) helps assess the variability of the results by training and evaluating the model on multiple random splits of the data.
* **Training Set Size:** This option allows you to manually specify the size (percentage) of the training set. Leaving it blank or at 100% uses all data for training, which means no separate testing set is created in this case (not recommended for evaluation).

**4. kNN and Distance Widget:**

* the Distance widget isn't directly connected to the kNN widget in Orange. However, the kNN algorithm does implicitly calculate distances internally.
* When you provide preprocessed data to the workflow, the features (columns) represent the characteristics of the data points (wine samples). The kNN model calculates distances between these data points based on their feature values to identify nearest neighbors.

**Additional Tips:**

* **Cross-Validation:** Consider using more folds (e.g., 5 or 10) for a more robust evaluation of your kNN model.
* **Hyperparameter Tuning:** Experiment with different k values (number of nearest neighbors) in the kNN widget to potentially improve model performance.
* **Data Visualization:** Explore the data using scatter plots or other visualizations to understand potential relationships between features.

In k-Nearest Neighbors (kNN), there's no explicit training process in the traditional sense of machine learning models that learn weights or parameters. Instead, kNN focuses on memorizing the training data.

Here's a breakdown of training and testing in kNN:

**Training Process:**

1. **Data Preparation:** You provide preprocessed data to the kNN algorithm. This data includes data points (e.g., wine samples) represented by features (columns) like color, aroma, etc.
2. **No Parameter Learning:** kNN doesn't learn or adjust internal parameters during training. It simply stores the entire training data in memory.

**Testing Process:**

1. **New Data Point:** You present a new, unseen data point (e.g., a new wine sample) to the kNN model.
2. **Distance Calculation:** kNN calculates the distances between the new data point and **all** data points in the **training set**. It uses a distance metric like Euclidean distance to measure similarity based on feature values.
3. **Nearest Neighbors:** Based on the calculated distances, kNN identifies the **k** nearest neighbors (most similar data points) to the new data point. The value of k (number of neighbors) is a hyperparameter you choose, influencing how localized the prediction will be.
4. **Prediction:**
   * **Classification (if applicable):** If your data has class labels (e.g., wine types), kNN predicts the class label that appears most frequently among the k nearest neighbors (majority vote).
   * **Regression (if applicable):** If your data is continuous (e.g., predicting wine quality score), kNN predicts the average value of the target variable (e.g., quality score) among the k nearest neighbors.

Here are some key points to remember:

* kNN memorizes the training data, making it fast to predict for new data points.
* There's no parameter learning, so kNN is not susceptible to overfitting in the usual sense.
* The choice of k and distance metric can significantly impact the model's performance. Experiment with different values to find the best configuration for your data.

**Key Takeaways:**

* kNN doesn't learn parameters in the traditional sense. It memorizes the training data.
* During testing, kNN predicts based on the closest neighbors in the training set.

**E.g., we see the following metrics data in Test and Scoring widget:**Train Test AUC CA F1 Prec MCC Spec LogLoss

0.034 0.02 0.98 0.96 0.96 0.96 0.94 0.98 0.82

1. **Train:**
   * This refers to the performance of the kNN model on the training dataset. It's a measure of how well the model fits the training data.
   * In your case, the value is 0.034, indicating that the kNN model performed well on the training set.
2. **Test:**
   * This refers to the performance of the kNN model on a separate testing dataset (not used during training). It gives an indication of how well the model generalizes to new, unseen data.
   * The value of 0.02 suggests that the kNN model performed slightly worse on the test set compared to the training set.
3. **AUC (Area Under the ROC Curve):**
   * AUC is a measure of the model's ability to distinguish between positive and negative classes. It ranges from 0 to 1, where a higher value indicates better performance.
   * Your AUC value of 0.98 is quite high, suggesting that the kNN model has a strong discriminatory power.
4. **CA (Classification Accuracy):**
   * This is the ratio of correctly predicted instances to the total instances. It provides a general measure of model accuracy.
   * A value of 0.96 indicates that the kNN model correctly classified 96% of instances in both the training and testing datasets.
5. **F1 Score:**
   * The F1 score is the harmonic mean of precision and recall. It provides a balance between precision and recall.
   * Your F1 score of 0.96 is high, indicating a good balance between precision and recall.
6. **Precision:**
   * Precision measures the accuracy of positive predictions. It is the ratio of true positive predictions to the total positive predictions.
   * A precision value of 0.96 suggests that the kNN model has a high accuracy in predicting positive instances.
7. **MCC (Matthews Correlation Coefficient):**
   * MCC is a correlation coefficient between the observed and predicted binary classifications. It considers true and false positives and negatives.
   * A value of 0.94 indicates a strong correlation between predicted and observed classifications.
8. **Spec (Specificity):**
   * Specificity measures the ability of the model to correctly identify negative instances.
   * A value of 0.98 suggests that the kNN model has a high specificity, meaning it is good at identifying true negatives.
9. **LogLoss:**
   * LogLoss is a measure of how well the predicted probabilities match the true labels. Lower values are better.
   * Your LogLoss value of 0.82 is reasonable, and lower values would indicate better-calibrated probability predictions.

**Summary,** this kNN model performs well, especially in terms of AUC, CA, F1, Precision, MCC, and Specificity. The LogLoss value is also acceptable. The slight drop in performance from training to testing data suggests a moderate level of generalization

**MNIST Case**, where we have 0..- 9 outputs/classes. How are all those metrics intrepreted for MNIST case.:

1. **True Positives (TP):**
   * In a multi-class setting like MNIST, TP refers to the instances where the model correctly predicts a specific digit among the 10 possible digits.
2. **False Negatives (FN):**
   * FN occurs when the model fails to predict a certain digit, which is actually present in the true labels.
3. **True Negatives (TN):**
   * In multi-class problems, TN is less applicable because it represents instances that are correctly predicted as not belonging to a specific class. It's more commonly used in binary classification.
4. **False Positives (FP):**
   * Similarly, FP in a multi-class context refers to instances that are incorrectly predicted as belonging to a specific class when they do not.
5. **Precision:**
   * In the MNIST case, precision for a specific digit measures how many instances predicted as that digit are actually that digit. It's the ratio of true positives to the sum of true positives and false positives for that digit.
6. **Recall (Sensitivity):**
   * Recall for a specific digit measures how many instances of that digit were correctly predicted out of all instances that are actually that digit. It's the ratio of true positives to the sum of true positives and false negatives for that digit.
7. **F1 Score:**
   * The F1 score is the harmonic mean of precision and recall for a specific digit. It provides a balanced measure of how well the model performs for that particular digit.
8. **Accuracy:**
   * Accuracy in a multi-class setting is the overall correctness of predictions across all classes. It's the ratio of correct predictions (true positives and true negatives for all classes) to the total number of instances.
9. **Macro/Micro-Average:**
   * When dealing with multiple classes, you might see macro and micro-average metrics. Micro-average considers all instances and classes equally, while macro-average calculates metrics for each class independently and then takes the average.
10. **AUC (Area Under the ROC Curve):**

* In a multi-class context, AUC is often calculated using the one-vs-rest (OvR) approach. It measures the ability of the model to distinguish between one class and the rest.

1. **LogLoss:**

* LogLoss is a measure of how well the predicted probabilities match the true labels across all classes. Lower values indicate better-calibrated probability predictions.

In summary, when evaluating a multi-class classification model like the one for the MNIST dataset, it's important to consider precision, recall, F1 score, accuracy, and potentially AUC and LogLoss, with specific attention to each digit's performance in addition to overall model performance.

**Confusion Matrix in Multi-Class Classification**

A confusion matrix is a powerful tool used to evaluate the performance of a classification model, especially when dealing with multiple classes. It provides a clear visualization of how the model classified the data points and allows you to identify areas for improvement.

**Structure:**

* The confusion matrix is a square table with rows representing the **actual classes** of the data points and columns representing the **predicted classes** by the model.
* Each cell of the table contains the number of data points that belong to a particular **actual class** (row) but were **predicted** by the model as belonging to a different **predicted class** (column).

**Interpreting the Matrix:**

* **Perfect Classification:** If the model perfectly classified all data points, the diagonal elements (where the actual class and predicted class match) would contain all the values, and other cells would be zero.
* **Misclassifications:** Off-diagonal elements represent **misclassifications**. The higher the value in a non-diagonal cell, the more data points were incorrectly classified into that category.

**Key Metrics Derived from Confusion Matrix:**

* **Accuracy:** Overall percentage of correctly classified data points.
* **Precision:** Ratio of correctly predicted positive cases to the total predicted positive cases. (For class i: Precision\_i = TP\_i / (TP\_i + FP\_i))
* **Recall:** Ratio of correctly predicted positive cases to the total actual positive cases. (For class i: Recall\_i = TP\_i / (TP\_i + FN\_i))
* **F1-Score:** Harmonic mean of precision and recall, combining their importance. (For class i: F1\_i = 2 \* (Precision\_i \* Recall\_i) / (Precision\_i + Recall\_i))

**Confusion Matrix Example (3 Classes):**

|  |  |  |  |
| --- | --- | --- | --- |
| Predicted Class | Class A | Class B | Class C |
| **Class A** | **TP\_A** (Correct) | FP\_B | FP\_C |
| **Class B** | FN\_A | **TP\_B** (Correct) | FP\_C |
| **Class C** | FN\_A | FN\_B | **TP\_C** (Correct) |

**Benefits of Confusion Matrix:**

* **Visualization:** Provides a clear visual representation of classification performance.
* **Identification of Errors:** Helps pinpoint specific classes where the model struggles, allowing for targeted improvement strategies.
* **Evaluation of Multi-Class Models:** More informative than just accuracy for multi-class problems, as it reveals issues in specific class predictions.

**Additional Considerations:**

* When dealing with imbalanced datasets (unequal distribution of classes), accuracy alone might not be sufficient. Analyzing metrics like precision, recall, and F1-score for each class is crucial.
* Confusion matrix interpretation should be done in conjunction with other evaluation metrics to gain a comprehensive understanding of the model's performance.

By utilizing the confusion matrix and the derived metrics, you can effectively assess the strengths and weaknesses of your multi-class classification model and take steps to improve its performance.

1. **Basic Structure**: A confusion matrix is essentially a table where each row represents the instances in an actual class while each column represents the instances in a predicted class (or vice versa). The diagonal elements of the matrix represent the number of instances that were correctly classified, while off-diagonal elements represent misclassifications.
2. **True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN)**: These are the four basic elements of a confusion matrix:
   * True Positives (TP): The number of instances correctly predicted as belonging to the positive class.
   * True Negatives (TN): The number of instances correctly predicted as not belonging to the positive class.
   * False Positives (FP): Also known as Type I errors, these are instances incorrectly predicted as belonging to the positive class when they actually belong to the negative class.
   * False Negatives (FN): Also known as Type II errors, these are instances incorrectly predicted as not belonging to the positive class when they actually belong to the positive class.
3. **Interpretation**: The confusion matrix allows you to gain insights into the performance of your classifier. By examining the values in the matrix, you can assess how well your model is performing in terms of correctly and incorrectly classifying instances across different classes.
4. **Metrics Derivation**: Various evaluation metrics can be derived from the confusion matrix, including:
   * Accuracy: (TP + TN) / Total
   * Precision: TP / (TP + FP)
   * Recall (Sensitivity or True Positive Rate): TP / (TP + FN)
   * Specificity (True Negative Rate): TN / (TN + FP)
   * F1 Score: 2 \* (Precision \* Recall) / (Precision + Recall)
5. **Visualization**: Confusion matrices can also be visualized graphically, making it easier to interpret the results, especially when dealing with multiple classes.

In summary, a confusion matrix is a crucial tool for evaluating the performance of a classification model, especially in scenarios involving multiple classes, by providing detailed insights into the model's predictive capabilities across different categories.

**Approaches to Validation:**

1. *Usual*: S= S\_t + S\_v (t – training, v – validation)

2. *k-Fold Cross Validation*:

In k-fold cross validation the original training set is partitioned into k subsets (folds) of size m/k (for simplicity, assume that k1=m/k is an integer). For each fold, the algorithm is trained on the union of the other folds and then the error of its

output is estimated using the fold. Finally, the average of all these errors is the estimate of the true error.

*The special case k = m, where m is the number of examples, is called leave-one-out (LOO).*

3. *Train-Validation-Test* split : S= S\_t + S\_v + S\_t

In most practical applications, we split the available examples into three sets. The 1st set is used for training our algorithm and the second is used as a validation set for model selection. After we select the best model, we test the performance of the output predictor on the *third* set, which is often called the *test set*."

The number obtained is used as an estimator of the true error of the learned predictor.

Briefly to subsequent Labs:  
**Dimensionality reduction methods**

**PCA meth**

Let x1, : : : , xm be *m* vectors in R^d We would like to reduce the dimensionality of these vectors using a linear transformation.

A matrix W \in R^(n,d), where n < d, induces a mapping x 🡪 Wx, where Wx \in R^n is the lower dimensionality

representation of x. Then, a second matrix U \in R^d;n can be used to (approximately) recover each original vector *x* from its compressed version. That is, for a compressed vector y = Wx, where y is in the low dimensional space Rn, we can construct ~x = Uy, so that ~x is the recovered version of x and resides in the

original high dimensional space R^d. In PCA, we find the compression matrix W and the recovering matrix U so that the total squared distance between the original and recovered vectors is minimal; namely, we aim at solving the problem

 (23.1)







**Random Projections**

Let x1, x2 be two vectors in Rd. A matrix W does not distort too much the distance between x1 and x2 if the ratio



is close to 1.

To show that ||Wx1 -Wx2|| is not too far away from ||x1 - x2|| is to show that W does not distort the norm of the vector

x = x1 - x2.





**Compressed Sensing** method

Compressed sensing is a dimensionality reduction technique which utilizes a prior assumption that the original vector is sparse in some basis. To motivate compressed sensing, consider a vector x \in Rd that has at most s nonzero elements

It is helpful \_first to understand when each of the methods can guarantee perfect recovery. PCA guarantees perfect recovery whenever the set of examples is contained in an n dimensional subspace of Rd. Compressed sensing guarantees perfect recovery whenever the set of examples is sparse (in some basis). On the basis of these observations, we can describe cases in which PCA will be better than compressed sensing and vice versa.

**Summary**

Two methods for dimensionality reduction using linear transformations: PCA and random projections.

PCA is optimal in the sense of averaged squared reconstruction error, if we restrict the reconstruction procedure to be linear as well. However, if we allow nonlinear reconstruction, PCA is not necessarily the optimal procedure. In particular, for sparse data, random projections can significantly outperform PCA. This fact is at the heart of the compressed sensing method.

**Comparison, Recommendations**

1. **Principal Component Analysis (PCA):**
   * **Algorithm:** PCA aims to transform the data into a new coordinate system, where the maximum variance lies along the principal components. It does this by finding the eigenvectors and eigenvalues of the covariance matrix.
   * **Features:**
     + Captures most of the variance in the data.
     + Principal components are orthogonal, enabling decorrelation.
   * **Advantages:**
     + Widely used and well-understood.
     + Effective for linearly correlated features.
   * **Disadvantages:**
     + Assumes linear relationships.
     + May not perform well on non-linear datasets.
2. **Random Projections:**
   * **Algorithm:** Random projections involve transforming the data by multiplying it with a random matrix. Despite being random, these projections can preserve pairwise distances reasonably well.
   * **Features:**
     + Faster than PCA and suitable for large datasets.
     + Simple and efficient.
   * **Advantages:**
     + Computational efficiency.
     + Preserves distances in high-dimensional spaces.
   * **Disadvantages:**
     + May not perform as well as PCA on low-dimensional data.
3. **Compressed Sensing:**
   * **Algorithm:** Compressed sensing aims to reconstruct a signal from a small set of linear measurements. It involves solving an optimization problem to find the sparsest solution.
   * **Features:**
     + Effective for sparse data.
     + Useful in situations where measurements are expensive or limited.
   * **Advantages:**
     + Handles sparse data well.
     + Useful for scenarios with limited measurements.
   * **Disadvantages:**
     + Requires sparsity assumption.
     + Can be computationally intensive.

**Appliocation**

**Principal Component Analysis (PCA)** is a versatile technique that can be applied to various types of problems across different domains. Here are some key classes of problems where PCA is commonly implemented:

1. **Clustering Problems:**
   * **K-Means Clustering**: PCA can be used as a preprocessing step to reduce the dimensionality of data before applying K-Means clustering. By reducing the number of features, PCA can help improve the performance and efficiency of clustering algorithms.
   * **Hierarchical Clustering**: PCA can also be applied before hierarchical clustering methods to reduce the computational complexity and improve clustering results.
2. **Classification Problems:**
   * **Multi-Class Classification**: PCA can be used to reduce the dimensionality of feature space in multi-class classification problems. By reducing the number of features, PCA can help simplify the classification task and improve the performance of classifiers.
   * **Imbalanced Classification**: In imbalanced classification problems where the number of samples in each class is not balanced, PCA can help address the class imbalance issue by reducing the dimensionality of feature space and improving the generalization performance of classifiers.
3. **Regression Problems:**
   * **Linear Regression**: PCA can be applied to preprocess data in linear regression problems to reduce the dimensionality of feature space and remove multicollinearity among predictor variables. This can help improve the interpretability and generalization performance of linear regression models.
   * **Non-linear Regression**: PCA can also be used in non-linear regression problems to preprocess data and extract the most informative features for predicting the target variable. By reducing the dimensionality of feature space, PCA can help alleviate overfitting and improve the accuracy of non-linear regression models.
4. **Neural Networks (N-Nets):**
   * **Dimensionality Reduction in Deep Learning**: PCA can be used as a dimensionality reduction technique in deep learning models to preprocess high-dimensional data before feeding it into neural networks. By reducing the dimensionality of input data, PCA can help speed up training and improve the convergence of neural networks.
5. **Other Directions:**
   * **Anomaly Detection**: PCA can be applied in anomaly detection problems to identify outliers or anomalous data points in high-dimensional feature space. By projecting data onto lower-dimensional subspaces, PCA can help detect deviations from the normal data distribution.
   * **Feature Extraction**: PCA can be used for feature extraction in various machine learning tasks, such as image recognition, signal processing, and natural language processing. By extracting the most informative features from high-dimensional data, PCA can help improve the performance and interpretability of machine learning models.

**Overall**, PCA is a versatile tool that can be applied to a wide range of problems in machine learning, data analysis, and pattern recognition. Its ability to reduce the dimensionality of data while preserving the most important information makes it a valuable technique in many applications.

**5. Other Methods:** (will be added later on)

t-SNE

SVM