Lecture 5: Preprocessing and sklearn pipelines

Firas Moosvi (Slides adapted from Varada Kolhatkar)

Announcements

- HW1 grades have been posted.
- Homework 1 solutions have been posted on Canvas under Files tab. Please do not share them with anyone or do not post them anywhere.

Recap from last class

Three slides left over from last class:

1. Curse of Dimensionality
2. SVMs with RBF kernel
3. Intuition of `C` and `gamma` in SVM RBF

Curse of dimensionality

- As dimensionality increases, the volume of the space increases exponentially, making the data sparse.
- Distance metrics lose meaning
  - Accidental similarity swamps out meaningful similarity
  - All points become almost equidistant.
- Overfitting becomes likely: Harder to generalize with high-dimensional data.
- How to deal with this?
  - Dimensionality reduction (PCA) (not covered in this course)
  - Feature selection techniques.

SVMs with RBF kernel

- RBF Kernel: Radial Basis Function, a way to transform data into higher dimensions implicitly.
- Strengths
  - Effective in high-dimensional and sparse data
  - Good performance on non-linear problems.
- Hyperparameters:
  - C$: Regularization parameter (trade-off between correct classification of training examples and maximization of the decision margin).
  - γ: Defines how far the influence of a single training example reaches.

Intuition of `C` and `gamma` in SVM RBF

- `C` (Regularization): Controls the trade-off between perfect training accuracy and having a simpler decision boundary.
  - High C: Strict, complex boundary (overfitting risk).
  - Low C: More errors allowed, smoother boundary (generalizes better).
- `Gamma` (Kernel Width): Controls the influence of individual data points.
  - High Gamma: Points have local impact, complex boundary.
  - Low Gamma: Points affect broader areas, smoother boundary.
- Key trade-off: Proper balance between `C` and `gamma` is crucial for avoiding overfitting or underfitting.

Recap

- Decision trees: Split data into subsets based on feature values to create decision rules
- k-NNs: Classify based on the majority vote from k nearest neighbors
- SVM RBFs: Create a boundary using an RBF kernel to separate classes

Synthesizing existing knowledge

Recap

Aspect	Decision Trees	K-Nearest Neighbors (KNN)	Support Vector Machines (SVM) with RBF Kernel
Main hyperparameters	Max depth, min samples split	Number of neighbors (k)	C (regularization), Gamma (RBF kernel width)
Interpretability			
Handling of Non-linearity			
Scalability			

Recap

Aspect	Decision Trees	K-Nearest Neighbors (KNN)	Support Vector Machines (SVM) with RBF Kernel
Sensitivity to Outliers			
Memory Usage			
Training Time			
Prediction Time			
Multiclass support			

UBC Computer Science

(iClicker) Exercise 5.1

iClicker cloud join link: **https://join.iclicker.com/VYFJ**

Take a guess: In your machine learning project, how much time will you typically spend on data preparation and transformation?

- a. ~80% of the project time
- b. ~20% of the project time
- c. ~50% of the project time
- d. None. Most of the time will be spent on model building

The question is adapted from here.

(iClicker) Exercise 5.2

iClicker cloud join link: **https://join.iclicker.com/VYFJ**

Select all of the following statements which are TRUE.

- a. StandardScaler ensures a fixed range (i.e., minimum and maximum values) for the features.
- b. StandardScaler calculates mean and standard deviation for each feature separately.
- c. In general, it’s a good idea to apply scaling on numeric features before training k-NN or SVM RBF models.
- d. The transformed feature values might be hard to interpret for humans.

After applying SimpleImputer The transformed data has a different shape than the original data.

(iClicker) Exercise 5.3

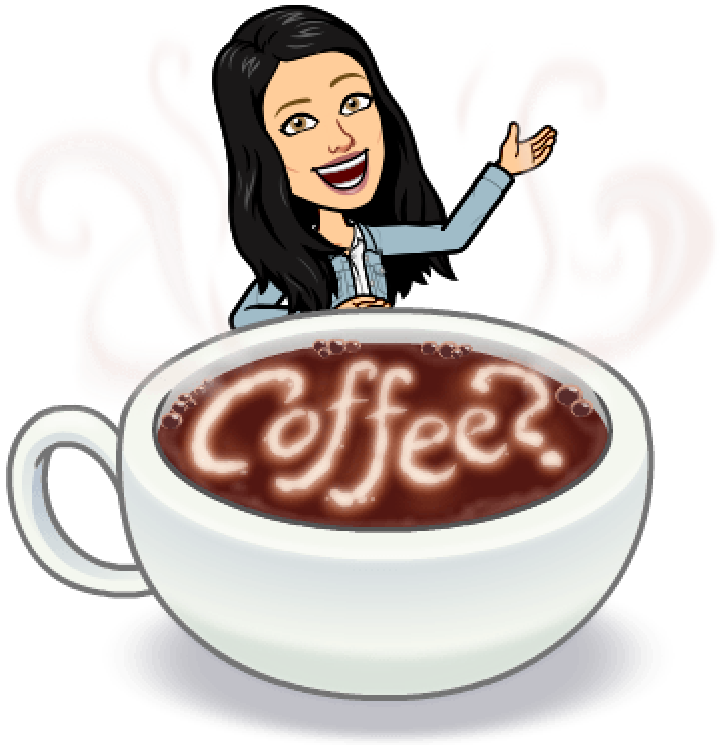
iClicker cloud join link: **https://join.iclicker.com/VYFJ**

Select all of the following statements which are TRUE.

- a. You can have scaling of numeric features, one-hot encoding of categorical features, and scikit-learn estimator within a single pipeline.
- b. Once you have a scikit-learn pipeline object with an estimator as the last step, you can call fit, predict, and score on it.
- c. You can carry out data splitting within scikit-learn pipeline.
- d. We have to be careful of the order we put each transformation and model in a pipeline.

Break

Let’s take a break!



/

Preprocessing motivation: example

You’re trying to find a suitable date based on:

- Age (closer to yours is better).
- Number of Facebook Friends (closer to your social circle is ideal).

Preprocessing motivation: example

- You are 30 years old and have 250 Facebook friends.

Person	Age	#FB Friends	Euclidean Distance Calculation	Distance
A	25	400	$\sqrt{(5^2 + 150^2)}$	150.08
B	27	300	$\sqrt{(3^2 + 50^2)}$	50.09
C	30	500	$\sqrt{(0^2 + 250^2)}$	250.00
D	60	250	$\sqrt{(30^2 + 0^2)}$	30.00

Based on the distances, the two nearest neighbors (2-NN) are:

- **Person D** (Distance: 30.00)
- **Person B** (Distance: 50.09)

What’s the problem here?

Common transformations

Imputation: Fill the gaps! (🟩🟧🟦)

Fill in missing data using a chosen strategy:

- **Mean**: Replace missing values with the average of the available data.
- **Median**: Use the middle value.
- **Most Frequent**: Use the most common value (mode).
- **KNN Imputation**: Fill based on similar neighbors.

Example:

Fill in missing values like filling empty seats in a classroom with the average student.

```python
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)
```

Scaling: Everything to the same range! (📉 📈)

Ensure all features have a comparable range.

- **StandardScaler**: Mean = 0, Standard Deviation = 1.
- **MinMaxScaler**: Scales features to a [0, 1] range.
- **RobustScaler**: Scales features using median and quantiles.

Example:

Rescaling everyone’s height to make basketball players and gymnasts comparable.

```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

One-Hot encoding: 🍎 → 1️⃣ 0️⃣ 0️⃣

Convert categorical features into binary columns.

- Creates new binary columns for each category.
- Useful for handling categorical data in machine learning models.

Example:

Turn “Apple, Banana, Orange” into binary columns:

Fruit	🍎	🍌	🍊
Apple 🍎	1	0	0
Banana 🍌	0	1	0
Orange 🍊	0	0	1

```python
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder()
X_encoded = encoder.fit_transform(X)
```

Ordinal encoding: Ranking matters! (⭐⭐⭐⭐⭐ → 1️⃣)

Convert categories into integer values that have a meaningful order.

- Assign integers based on order or rank.
- Useful when there is an inherent ranking in the data.

Example:

Turn “Poor, Average, Good” into 1, 2, 3:

Rating	Ordinal
Poor	1
Average	2
Good	3

```python
from sklearn.preprocessing import OrdinalEncoder
encoder = OrdinalEncoder()
X_ordinal = encoder.fit_transform(X)
```

`sklearn` Transformers vs Estimators

Transformers

- Are used to transform or preprocess data.
- Implement the `fit` and `transform` methods.
  - `fit(X)`: Learns parameters from the data.
  - `transform(X)`: Applies the learned transformation to the data.
- **Examples**:
  - **Imputation** (`SimpleImputer`): Fills missing values.
  - **Scaling** (`StandardScaler`): Standardizes features.

Estimators

- Used to make predictions.
- Implement `fit` and `predict` methods.
  - `fit(X, y)`: Learns from labeled data.
  - `predict(X)`: Makes predictions on new data.
- Examples: `DecisionTreeClassifier`, `SVC`, `KNeighborsClassifier`

```python
from sklearn.tree import DecisionTreeClassifier
tree_clf = DecisionTreeClassifier()
```

The golden rule in feature transformations

- **Never** transform the entire dataset at once!
- **Why**? It leads to **data leakage** — using information from the test set in your training process, which can artificially inflate model performance.
- **Fit** transformers like scalers and imputers on the **training set only**.
- **Apply** the transformations to both the training and test sets **separately**.

Example:

```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

`sklearn` Pipelines

- Pipeline is a way to chain multiple steps (e.g., preprocessing + model fitting) into a single workflow.
- Simplify the code and improves readability.
- Reduce the risk of data leakage by ensuring proper transformation of the training and test sets.
- Automatically apply transformations in sequence.

Example:

Chaining a `StandardScaler` with a `KNeighborsClassifier` model.

```python
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier

pipeline = make_pipeline(StandardScaler(), KNeighborsClassifier())

pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)
```

See you next week!