

Data preprocessing and Data Augmentation

Naeemullah Khan

naeemullah.khan@kaust.edu.sa



جامعة الملك عبد الله
للغعلوم والتكنولوجيا
King Abdullah University of
Science and Technology

KAUST Academy
King Abdullah University of Science and Technology

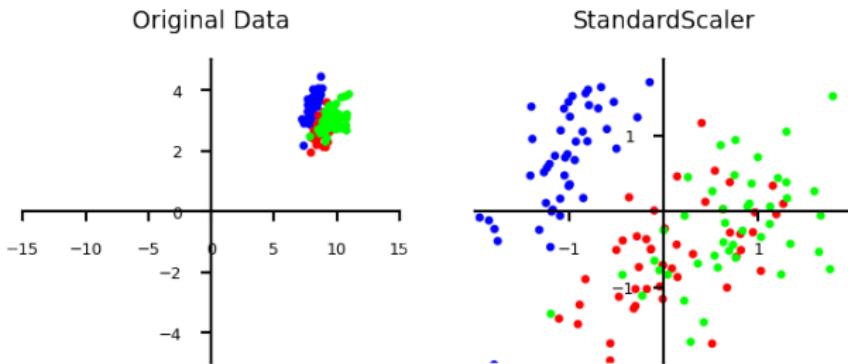
July 25, 2025

- ▶ Machine learning models make a lot of assumptions about the data
- ▶ In reality, these assumptions are often violated
- ▶ We build pipelines that transform the data before feeding it to the learners
 - Scaling (or other numeric transformations)
 - Encoding (convert categorical features into numerical ones)
 - Automatic feature selection
 - Feature engineering (e.g. binning, polynomial features,...)
 - Handling missing data
 - Handling imbalanced data
 - Dimensionality reduction (e.g. PCA)
 - Learned embeddings (e.g. for text)
- ▶ Seek the best combinations of transformations and learning methods

Data transformations (cont.)

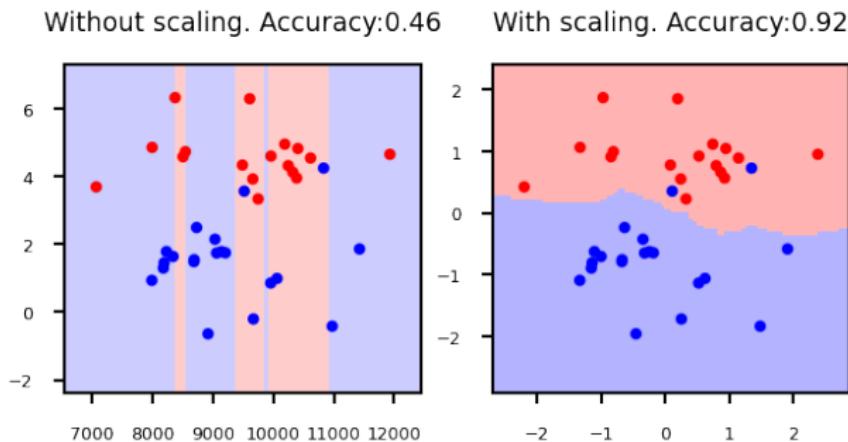
- Often done empirically, using cross-validation
- Make sure that there is no data leakage during this process!

- ▶ Use when different numeric features have different scales (different range of values)
 - Features with much higher values may overpower the others
- ▶ Goal: bring them all within the same range
- ▶ Different methods exist



Why do we need scaling?

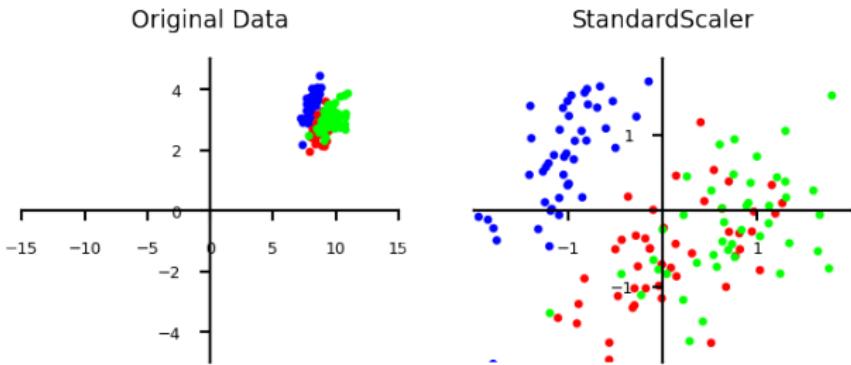
- ▶ KNN: Distances depend mainly on feature with larger values
- ▶ SVMs: (kernelized) dot products are also based on distances
- ▶ Linear model: Feature scale affects regularization
 - Weights have similar scales, more interpretable



Standard scaling (standardization)

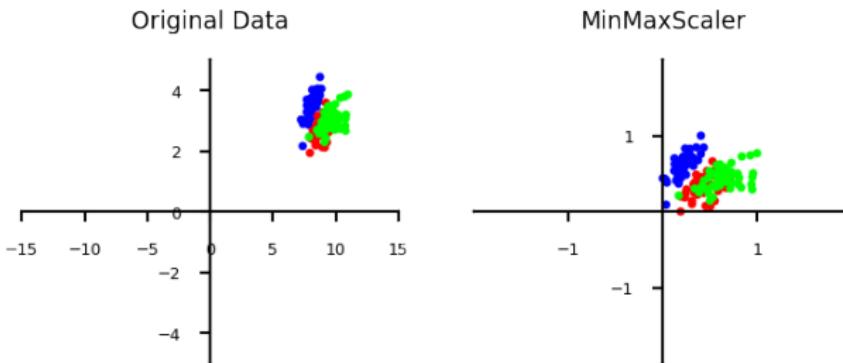
- ▶ Generally most useful, assumes data is more or less normally distributed
- ▶ Per feature, subtract the mean value μ , scale by standard deviation σ
- ▶ New feature has $\mu = 0$ and $\sigma = 1$, values can still be arbitrarily large

$$x_{\text{new}} = \frac{x - \mu}{\sigma}$$



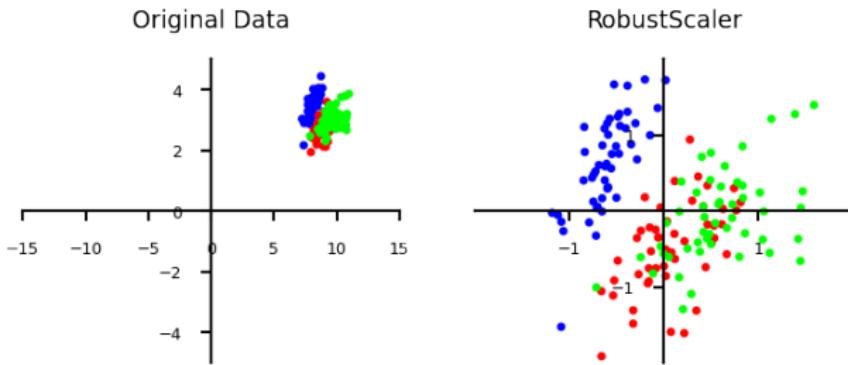
- ▶ Scales all features between a given *min* and *max* value (e.g. 0 and 1)
- ▶ Makes sense if min/max values have meaning in your data
- ▶ Sensitive to outliers

$$x_{\text{new}} = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}} \cdot (\text{max} - \text{min}) + \text{min}$$

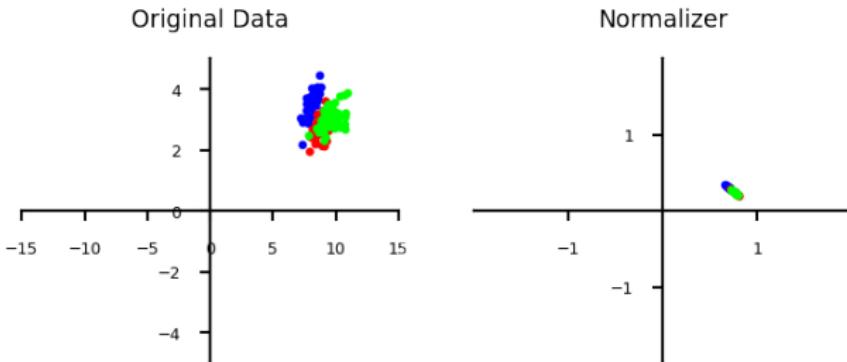


Robust scaling

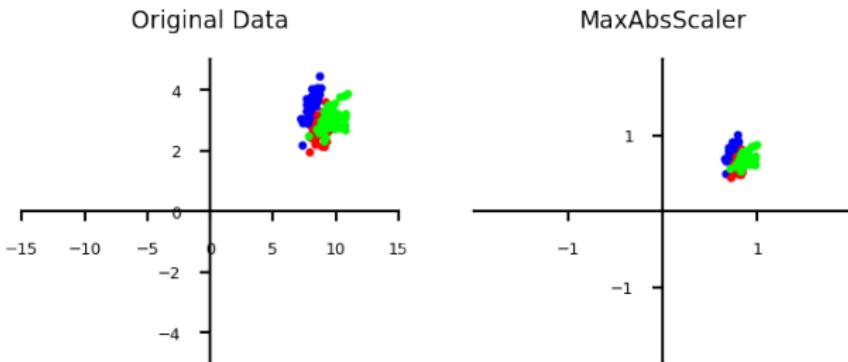
- ▶ Subtracts the median, scales between quantiles q_{25} and q_{75}
- ▶ New feature has median 0, $q_{25} = -1$ and $q_{75} = 1$
- ▶ Similar to standard scaler, but ignores outliers



- ▶ Makes sure that feature values of each point (each row) sum up to 1 (L1 norm)
 - Useful for count data (e.g. word counts in documents)
- ▶ Can also be used with L2 norm (sum of squares is 1)
 - Useful when computing distances in high dimensions
 - Normalized Euclidean distance is equivalent to cosine similarity

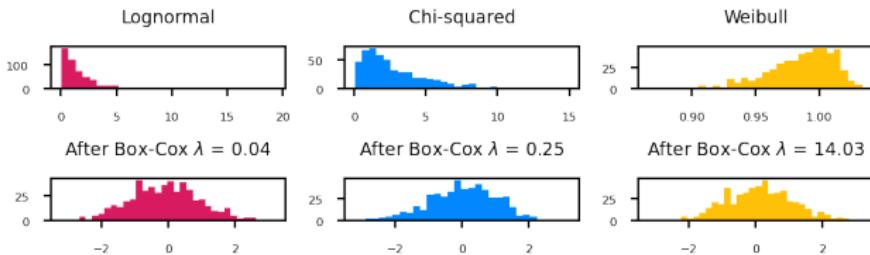


- ▶ For sparse data (many features, but few are non-zero)
 - Maintain sparseness (efficient storage)
- ▶ Scales all values so that maximum absolute value is 1
- ▶ Similar to Min-Max scaling without changing 0 values



- ▶ Some features follow certain distributions
 - E.g. number of twitter followers is log-normal distributed
- ▶ Box-Cox transformations transform these to normal distributions (λ is fitted)
 - Only works for positive values, use Yeo-Johnson otherwise

$$bc_\lambda(x) = \begin{cases} \log(x) & \lambda = 0 \\ \frac{x^\lambda - 1}{\lambda} & \lambda \neq 0 \end{cases}$$



Categorical feature encoding

- ▶ Many algorithms can only handle numeric features, so we need to encode the categorical ones

	boro	salary	vegan
0	Manhattan	103	0
1	Queens	89	0
2	Manhattan	142	0
3	Brooklyn	54	1
4	Brooklyn	63	1
5	Bronx	219	0

Ordinal encoding

- ▶ Simply assigns an integer value to each category in the order they are encountered
- ▶ Only really useful if there exist a natural order in categories
 - Model will consider one category to be 'higher' or 'closer' to another

	boro	boro_ordinal	salary
0	Manhattan	2	103
1	Queens	3	89
2	Manhattan	2	142
3	Brooklyn	1	54
4	Brooklyn	1	63
5	Bronx	0	219

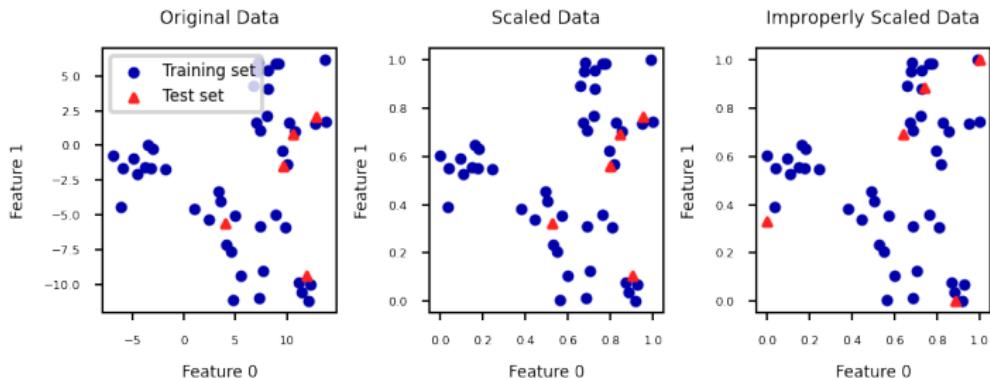
One-hot encoding (dummy encoding)

- ▶ Simply adds a new 0/1 feature for every category, having 1 (hot) if the sample has that category
- ▶ Can explode if a feature has lots of values, causing issues with high dimensionality
- ▶ What if test set contains a new category not seen in training data?
 - Either ignore it (just use all 0's in row), or handle manually (e.g. resample)

	boro	boro_Bronx	boro_Brooklyn	boro_Manhattan	boro_Queens	salary
0	Manhattan	0	0	1	0	103
1	Queens	0	0	0	1	89
2	Manhattan	0	0	1	0	142
3	Brooklyn	0	1	0	0	54
4	Brooklyn	0	1	0	0	63
5	Bronx	1	0	0	0	219

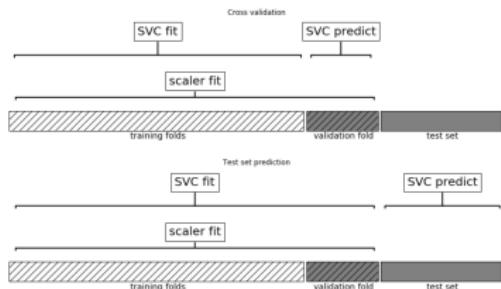
- ▶ Data transformations should always follow a fit-predict paradigm
 - Fit the transformer on the training data only
 - ▶ E.g. for a standard scaler: record the mean and standard deviation
 - Transform (e.g. scale) the training data, then train the learning model
 - Transform (e.g. scale) the test data, then evaluate the model
- ▶ Only scale the input features (X), not the targets (y)
- ▶ If you fit and transform the whole dataset before splitting, you get data leakage
 - You have looked at the test data before training the model
 - Model evaluations will be misleading
- ▶ If you fit and transform the training and test data separately, you distort the data
 - E.g. training and test points are scaled differently

- ▶ Properly scaled: fit on training set, transform on training and test set
- ▶ Improperly scaled: fit and transform on the training and test data separately
 - Test data points nowhere near same training data points

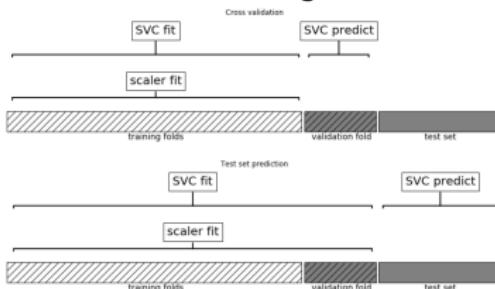


- ▶ Cross-validation: training set is split into training and validation sets for model selection
- ▶ Incorrect: Scaler is fit on whole training set before doing cross-validation
 - Data leaks from validation folds into training folds, selected model may be optimistic
- ▶ Right: Scaler is fit on training folds only

Information Leak



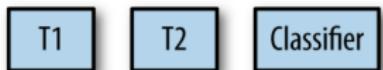
No Information leakage



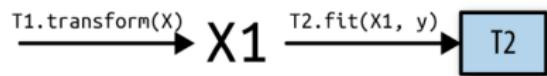
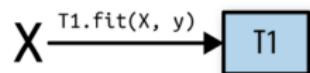
- ▶ A pipeline is a combination of data transformation and learning algorithms
- ▶ It has a `fit`, `predict`, and `score` method, just like any other learning algorithm
 - Ensures that data transformations are applied correctly

Pipelines (cont.)

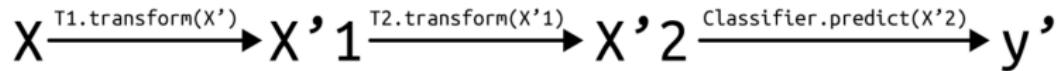
```
pipe = make_pipeline(T1(), T2(), Classifier())
```



```
pipe.fit(X, y)
```



```
pipe.predict(X')
```



In practice (scikit-learn)

- ▶ A pipeline combines multiple processing *steps* in a single estimator
- ▶ All but the last step should be data transformer (have a *transform* method)

```
# Make pipeline, step names will be 'minmaxscaler' and 'linearsvc'  
pipe = make_pipeline(MinMaxScaler(), LinearSVC())  
  
# Build pipeline with named steps  
pipe = Pipeline([("scaler", MinMaxScaler()), ("svm", LinearSVC())])  
  
# Correct fit and score  
score = pipe.fit(X_train, y_train).score(X_test, y_test)  
  
# Retrieve trained model by name  
svm = pipe.named_steps["svm"]  
  
# Correct cross-validation  
scores = cross_val_score(pipe, X, y)
```

- ▶ If you want to apply different preprocessors to different columns, use `ColumnTransformer`
- ▶ If you want to merge pipelines, you can use `FeatureUnion` to concatenate columns

```
# 2 sub-pipelines, one for numeric features, other for categorical ones
numeric_pipe = make_pipeline(SimpleImputer(), StandardScaler())
categorical_pipe = make_pipeline(SimpleImputer(), OneHotEncoder())

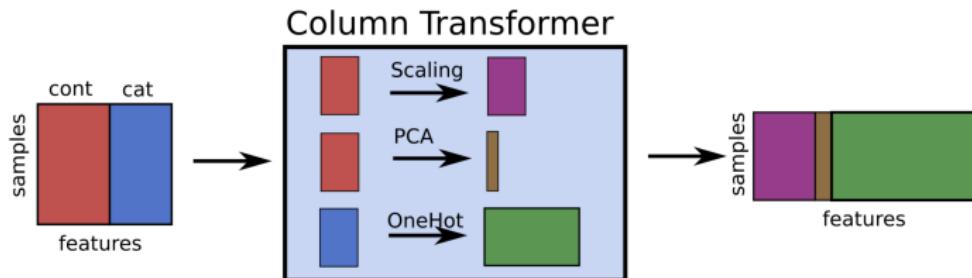
# Using categorical pipe for features A,B,C, numeric pipe otherwise
preprocessor = make_column_transformer((categorical_pipe,
                                       ["A", "B", "C"]),
                                       remainder=numeric_pipe)

# Combine with learning algorithm in another pipeline
pipe = make_pipeline(preprocessor, LinearSVC())

# Feature union of PCA features and selected features
union = FeatureUnion([("pca", PCA()), ("selected", SelectKBest())])
pipe = make_pipeline(union, LinearSVC())
```

- ▶ ColumnTransformer concatenates features in order

```
pipe = make_column_transformer((StandardScaler(), numeric_features),
                               (PCA(), numeric_features),
                               (OneHotEncoder(), categorical_features))
```



- ▶ We can safely use pipelines in model selection (e.g. grid search)
- ▶ Use `__` to refer to the hyperparameters of a step, e.g. `svm__C`

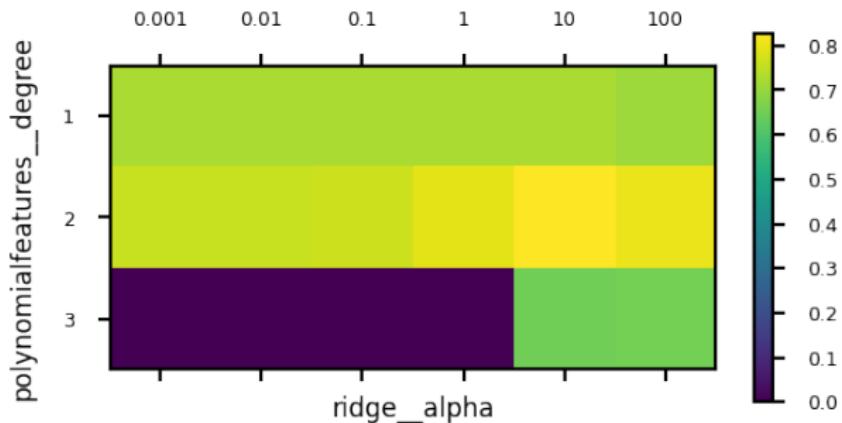
```
# Correct grid search (can have hyperparameters of any step)
param_grid = {'svm__C': [0.001, 0.01],
              'svm__gamma': [0.001, 0.01, 0.1, 1, 10, 100]}
grid = GridSearchCV(pipe, param_grid=param_grid).fit(X, y)

# Best estimator is now the best pipeline
best_pipe = grid.best_estimator_

# Tune pipeline and evaluate on held-out test set
grid = GridSearchCV(pipe, param_grid=param_grid).fit(X_train, y_train)
grid.score(X_test, y_test)
```

Example: Tune multiple steps at once

```
pipe = make_pipeline(StandardScaler(), PolynomialFeatures(), Ridge())
param_grid = {'polynomialfeatures__degree': [1, 2, 3],
              'ridge__alpha': [0.001, 0.01, 0.1, 1, 10, 100]}
grid = GridSearchCV(pipe, param_grid=param_grid).fit(X_train, y_train)
```

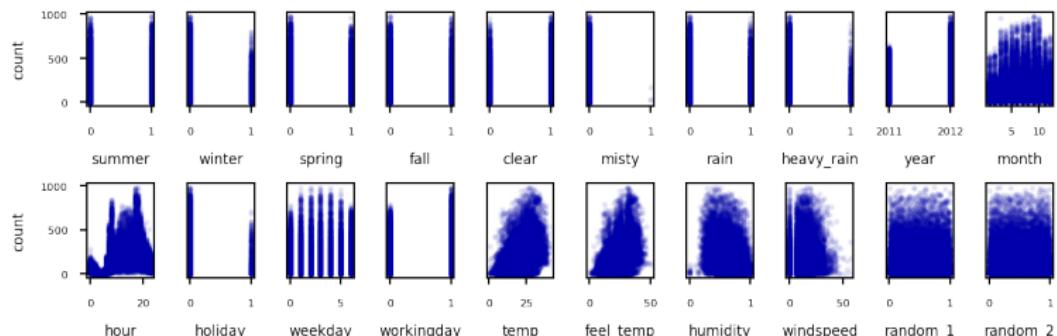


It can be a good idea to reduce the number of features to only the most useful ones

- ▶ Simpler models that generalize better (less overfitting)
 - Curse of dimensionality (e.g. kNN)
 - Even models such as RandomForest can benefit from this
 - Sometimes it is one of the main methods to improve models (e.g. gene expression data)
- ▶ Faster prediction and training
 - Training time can be quadratic (or cubic) in number of features
- ▶ Easier data collection, smaller models (less storage)
- ▶ More interpretable models: fewer features to look at

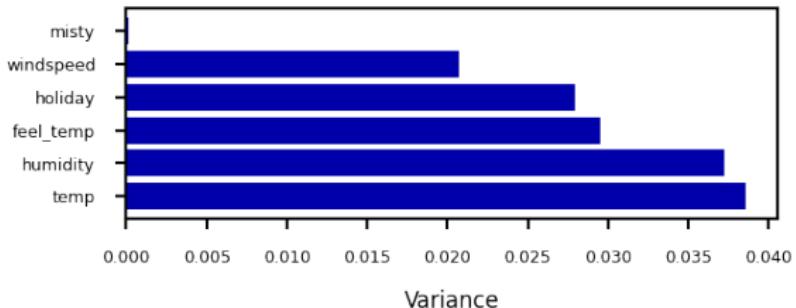
Example: bike sharing

- ▶ The Bike Sharing Demand dataset shows the amount of bikes rented in Washington DC
- ▶ Some features are clearly more informative than others (e.g. temp, hour)
- ▶ Some are correlated (e.g. temp and feel_temp)
- ▶ We add two random features at the end



- ▶ Variance-based
 - Remove (near) constant features
 - ▶ Choose a small variance threshold
 - Scale features before computing variance!
 - Infrequent values may still be important
- ▶ Covariance-based
 - Remove correlated features
 - The small differences may actually be important
 - ▶ You don't know because you don't consider the target

Unsupervised feature selection (cont.)

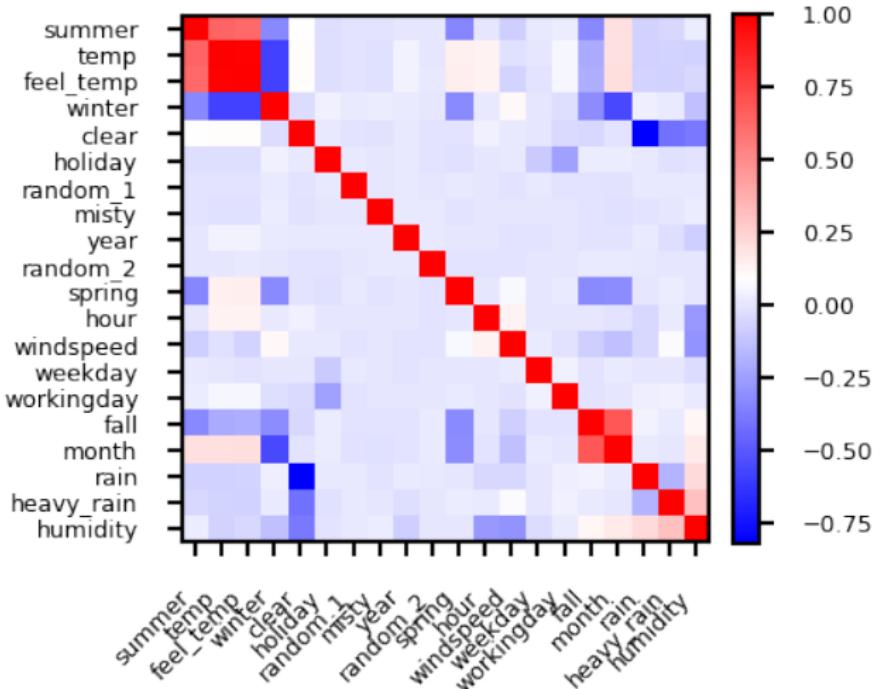


- ▶ Remove features X_i ($= \mathbf{X}_{\cdot,i}$) that are highly correlated (have high correlation coefficient ρ)

$$\rho(X_1, X_2) = \frac{\text{cov}(X_1, X_2)}{\sigma(X_1)\sigma(X_2)} = \frac{\frac{1}{N-1} \sum_i (X_{i,1} - \bar{X}_1)(X_{i,2} - \bar{X}_2)}{\sigma(X_1)\sigma(X_2)}$$

- ▶ Should we remove `feel_temp`? Or `temp`? Maybe one correlates more with the target?

Covariance based feature selection (cont.)

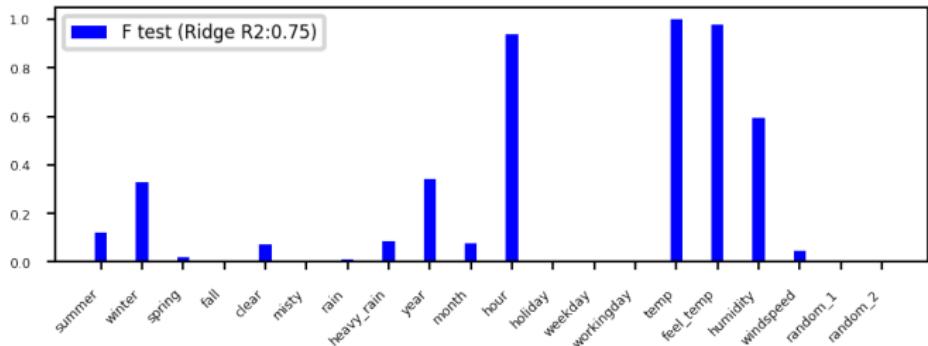


- ▶ Univariate: F-test and Mutual Information
- ▶ Model-based: Random Forests, Linear models, kNN
- ▶ Wrapping techniques (black-box search)
- ▶ Permutation importance

- ▶ Consider each feature individually (univariate), independent of the model that you aim to apply
- ▶ Use a statistical test: is there a *linear statistically significant relationship* with the target?
- ▶ Use F-statistic (or corresponding p value) to rank all features, then select features using a threshold
 - Best k , best $k\%$, probability of removing useful features (FPR), ...
- ▶ Cannot detect correlations (e.g. temp and feel_temp) or interactions (e.g. binary features)

Univariate statistics (F-test) (cont.)

Feature importance (selection threshold 0.75)



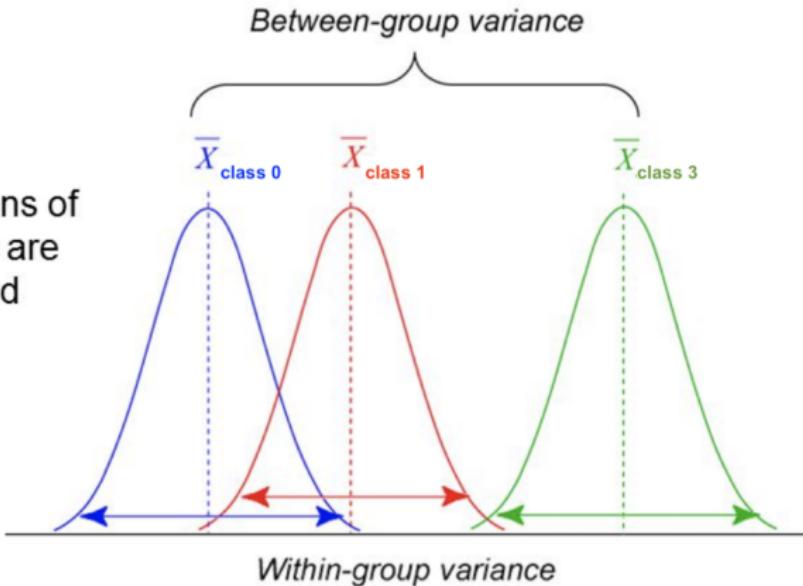
- ▶ For regression: does feature X_i correlate (positively or negatively) with the target y ?

$$\text{F-statistic} = \frac{\rho(X_i, y)^2}{1 - \rho(X_i, y)^2} \cdot (N - 1)$$

- ▶ For classification: uses ANOVA: does X_i explain the between-class variance?
 - Alternatively, use the χ^2 test (only for categorical features)

$$\text{F-statistic} = \frac{\text{within-class variance}}{\text{between-class variance}} = \frac{\text{var}(\bar{X}_i)}{\text{var}(X_i)}$$

The means of
3 groups are
compared

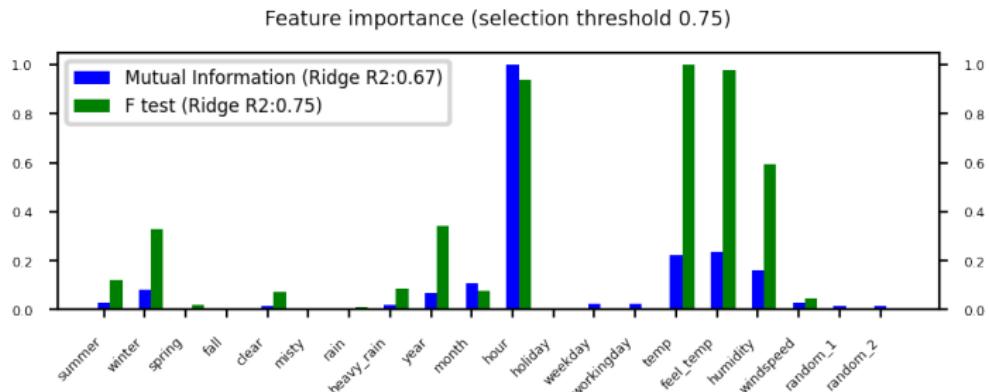


- ▶ Measures how much information X_i gives about the target Y . In terms of entropy H :

$$MI(X, Y) = H(X) + H(Y) - H(X, Y)$$

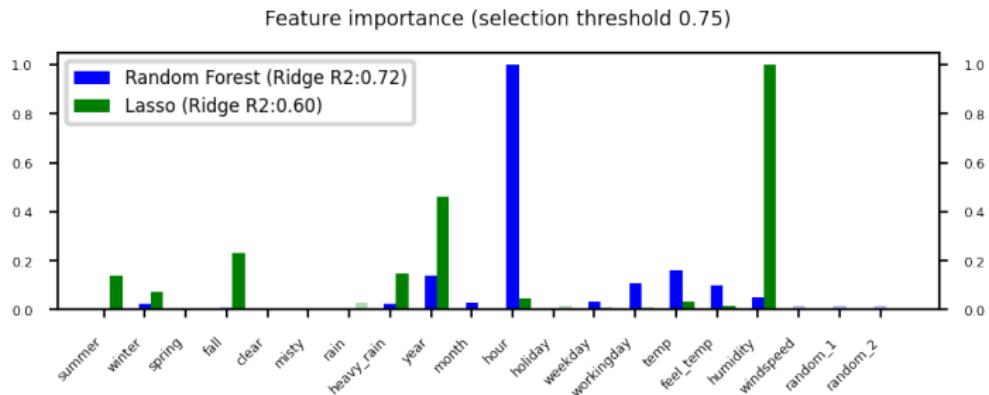
- ▶ Idea: estimate $H(X)$ as the average distance between a data point and its k Nearest Neighbors
 - You need to choose k and say which features are categorical
- ▶ Captures complex dependencies (e.g. hour, month), but requires more samples to be accurate

Mutual information (cont.)



- ▶ Use a **tuned(!)** supervised model to judge the importance of each feature
 - Linear models (Ridge, Lasso, LinearSVM, . . .): features with highest weights (coefficients)
 - Tree-based models: features used in first nodes (high information gain)
- ▶ Selection model can be different from the one you use for final modelling
- ▶ Captures interactions: features are more/less informative in combination (e.g. winter, temp)
- ▶ RandomForests: learns complex interactions (e.g. hour), but biased to high cardinality features

Model-based Feature Selection (cont.)

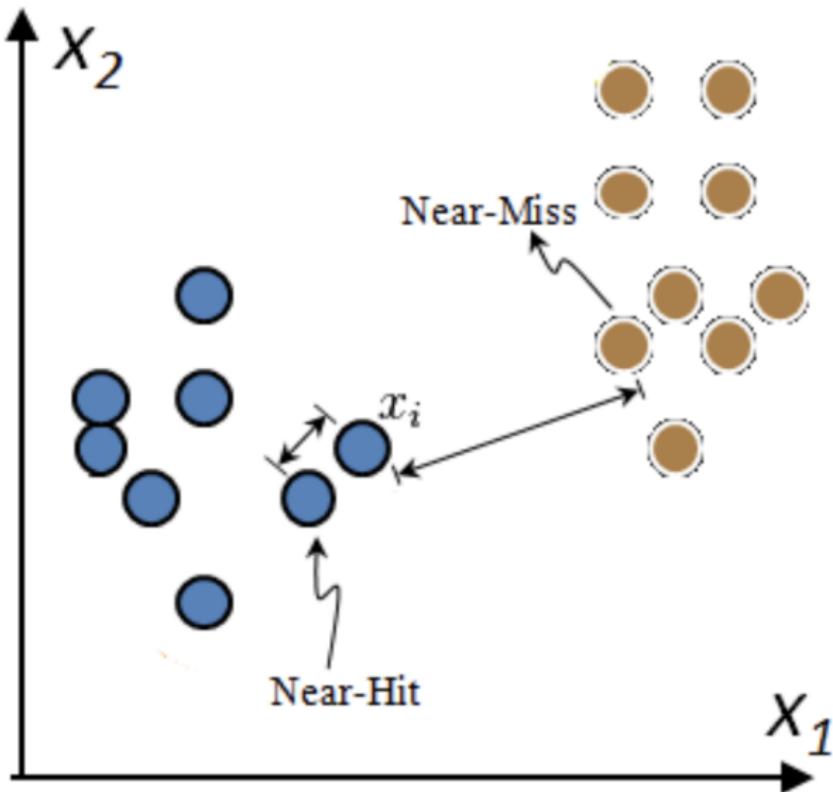


- ▶ For l iterations, choose a random point \mathbf{x}_i and find k nearest neighbors \mathbf{x}_k
- ▶ Increase feature weights if \mathbf{x}_i and \mathbf{x}_k have different class (near miss), else decrease

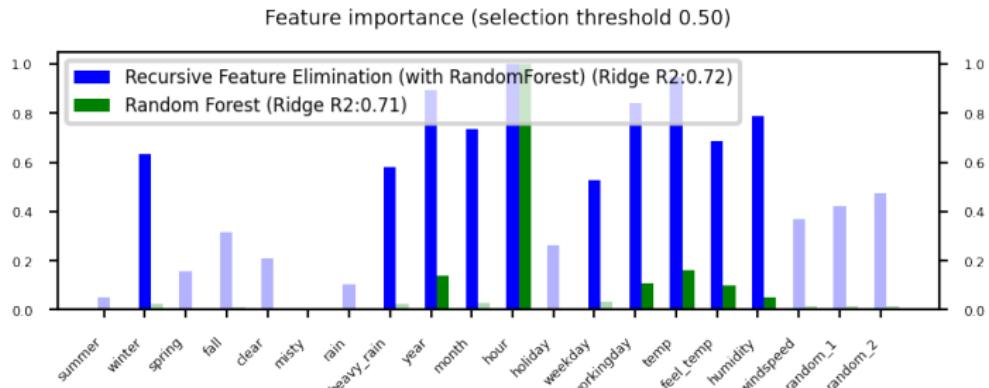
$$\mathbf{w}_i = \mathbf{w}_{i-1} + (\mathbf{x}_i - \text{nearMiss}_i)^2 - (\mathbf{x}_i - \text{nearHit}_i)^2$$

- ▶ Many variants: ReliefF (uses L1 norm, faster), RReliefF (for regression), ...

Relief: Model-based selection with kNN (cont.)

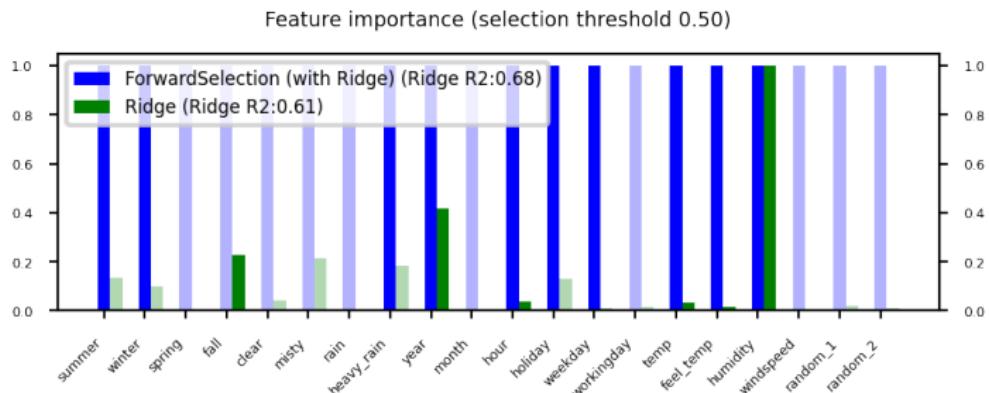


- ▶ Dropping many features at once is not ideal: feature importance may change in subset
- ▶ Recursive Feature Elimination (RFE)
 - Remove s least important feature(s), recompute remaining importances, repeat
- ▶ Can be rather slow



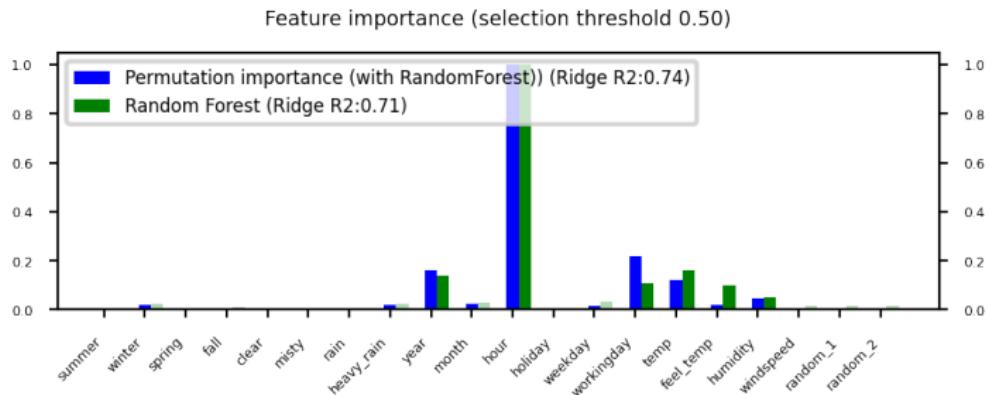
- ▶ Evaluate your model with different sets of features, find best subset based on performance
- ▶ Greedy black-box search (can end up in local minima)
 - Backward selection: remove least important feature, recompute importances, repeat
 - Forward selection: set aside most important feature, recompute importances, repeat
 - Floating: add best new feature, remove worst one, repeat (forward or backward)
- ▶ Stochastic search: use random mutations in candidate subset (e.g. simulated annealing)

Sequential feature selection (Wrapping) (cont.)

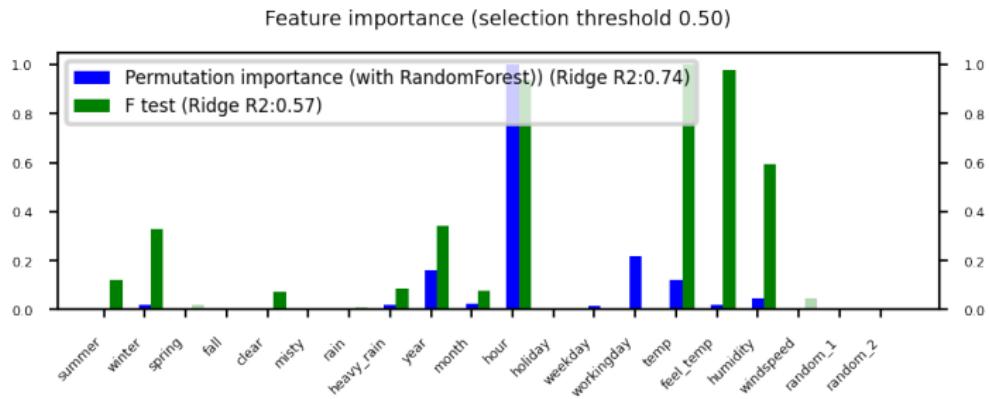


- ▶ Defined as the decrease in model performance when a single feature value is randomly shuffled
 - This breaks the relationship between the feature and the target
- ▶ Model agnostic, metric agnostic, and can be calculated many times with different permutations
- ▶ Can be applied to unseen data (not possible with model-based techniques)
- ▶ Less biased towards high-cardinality features (compared with RandomForests)

Permutation feature importance (cont.)



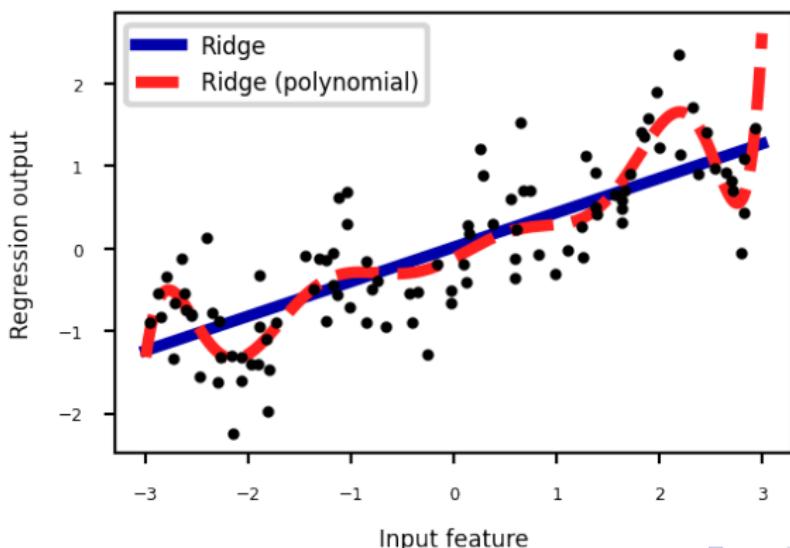
- ▶ Feature importances (scaled) and cross-validated R^2 score of pipeline
 - Pipeline contains feature selection + Ridge
- ▶ Selection threshold value ranges from 25% to 100% of all features
- ▶ Best method ultimately depends on the problem and dataset at hand



- ▶ Create new features based on existing ones
 - Polynomial features
 - Interaction features
 - Binning
- ▶ Mainly useful for simple models (e.g. linear models)
 - Other models can learn interactions themselves
 - But may be slower, less robust than linear models

- ▶ Add all polynomials up to degree d and all products
 - Equivalent to polynomial basis expansions

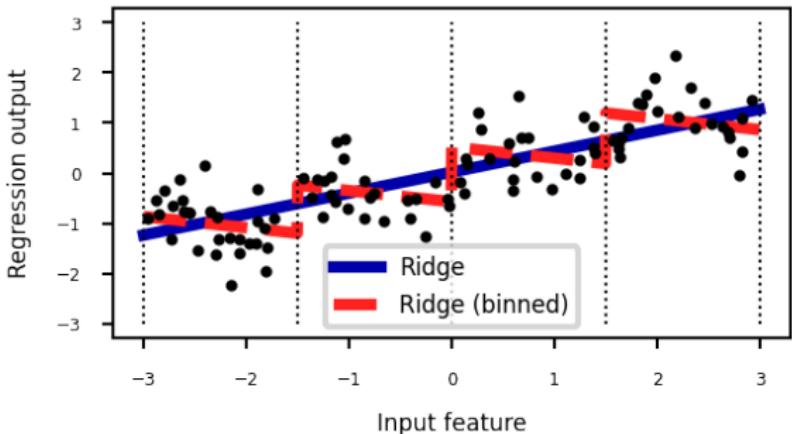
$$[1, x_1, \dots, x_p] \rightarrow [1, x_1, \dots, x_p, x_1^2, \dots, x_p^2, \dots, x_p^d, x_1x_2, \dots, x_{p-1}x_p]$$



- ▶ Partition numeric feature values into n intervals (bins)
- ▶ Create n new one-hot features, 1 if original value falls in corresponding bin
- ▶ Models different intervals differently (e.g. different age groups)

orig	[-3.0,-1.5]	[-1.5,0.0]	[0.0,1.5]	[1.5,3.0]
-0.752759	0.000000	1.000000	0.000000	0.000000
2.704286	0.000000	0.000000	0.000000	1.000000
1.391964	0.000000	0.000000	1.000000	0.000000

Binning (cont.)

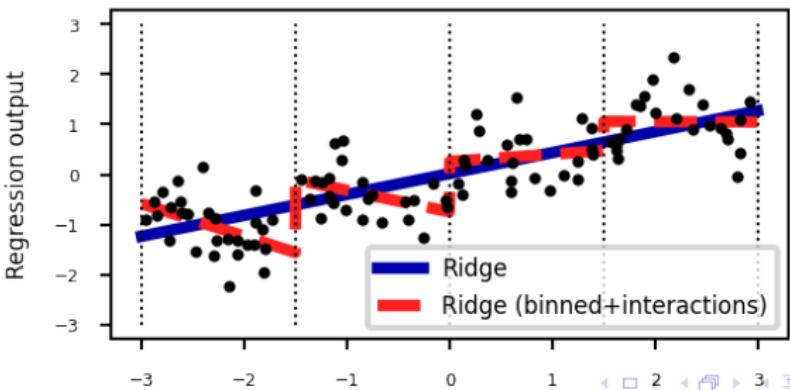


Binning + interaction features

► Add *interaction features* (or *product features*)

- Product of the bin encoding and the original feature value
- Learn different weights per bin

orig	b0	b1	b2	b3	X*b0	X*b1	X*b2	X*b3
-0.752759	0.000000	1.000000	0.000000	0.000000	-0.752759	-0.000000	-0.000000	-0.000000
2.704286	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	2.704286
1.391964	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	1.391964	0.000000



Categorical feature interactions

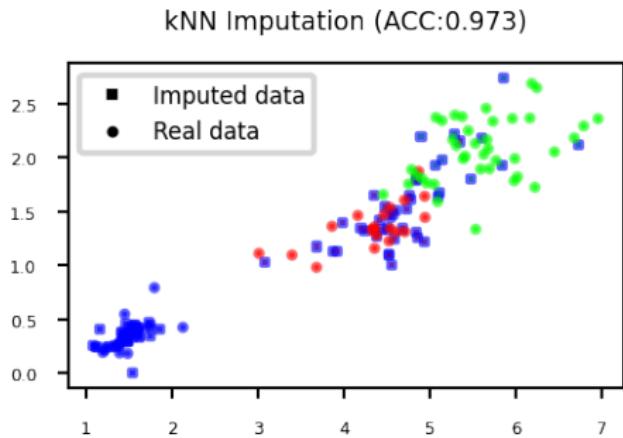
- ▶ One-hot-encode categorical feature
- ▶ Multiply every one-hot-encoded column with every numeric feature
- ▶ Allows to build different submodels for different categories

gender	age	pageviews	time
0 M	14	70	269
1 F	16	12	1522
2 M	12	42	235
3 F	25	64	63
4 F	22	93	21

age_M	pageviews_M	time_M	gender_M_M	gender_F_M	age_F	pageviews_F	time_F	gender_F_F
14	70	269	1	0	0	0	0	0
0	0	0	0	0	16	12	1522	1
12	42	235	1	0	0	0	0	0
0	0	0	0	0	25	64	63	1
0	0	0	0	0	22	93	21	1

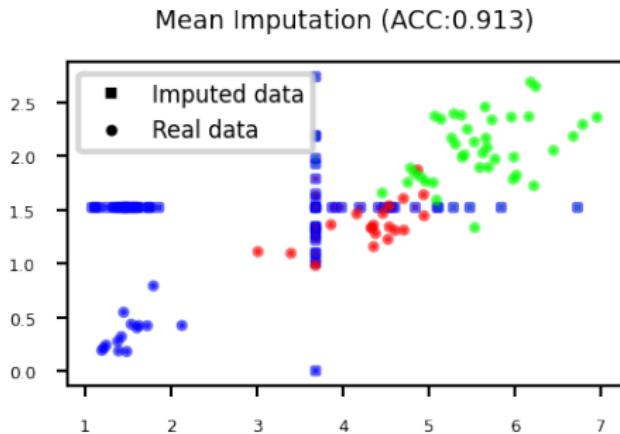
- ▶ Data can be missing in different ways:
 - Missing Completely at Random (MCAR): purely random points are missing
 - Missing at Random (MAR): something affects missingness, but no relation with the value
 - ▶ E.g. faulty sensors, some people don't fill out forms correctly
 - Missing Not At Random (MNAR): systematic missingness linked to the value
 - ▶ Has to be modelled or resolved (e.g. sensor decay, sick people leaving study)
- ▶ Missingness can be encoded in different ways: '?', '-1', 'unknown', 'NA', ...
- ▶ Also labels can be missing (remove example or use semi-supervised learning)

- ▶ Mean/constant imputation
- ▶ kNN-based imputation
- ▶ Iterative (model-based) imputation
- ▶ Matrix Factorization techniques

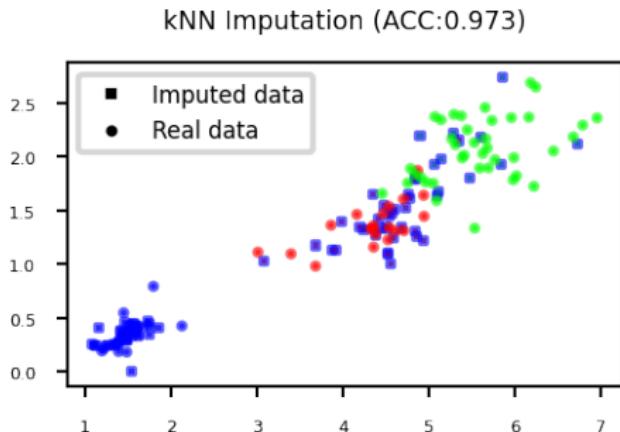


- ▶ Replace all missing values of a feature by the same value
 - Numerical features: mean or median
 - Categorical features: most frequent category
 - Constant value, e.g. 0 or 'missing' for text features
- ▶ Optional: add an indicator column for missingness
- ▶ Example: Iris dataset (randomly removed values in 3rd and 4th column)

Mean imputation (cont.)



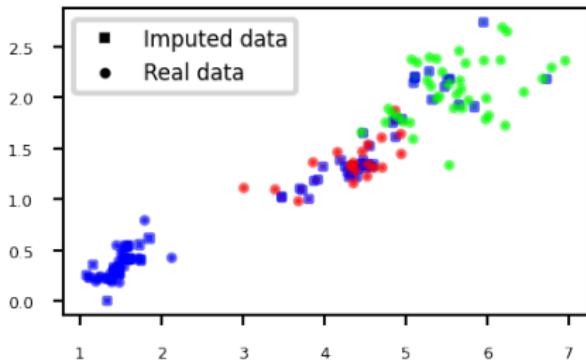
- ▶ Use special version of kNN to predict value of missing points
- ▶ Uses only non-missing data when computing distances



- ▶ Better known as Multiple Imputation by Chained Equations (MICE)
- ▶ Iterative approach
 - Do first imputation (e.g. mean imputation)
 - Train model (e.g. RandomForest) to predict missing values of a given feature
 - Train new model on imputed data to predict missing values of the next feature
 - ▶ Repeat m times in round-robin fashion, leave one feature out at a time

Iterative (model-based) Imputation (cont.)

Iterative Imputation (RandomForest) (ACC:0.960)

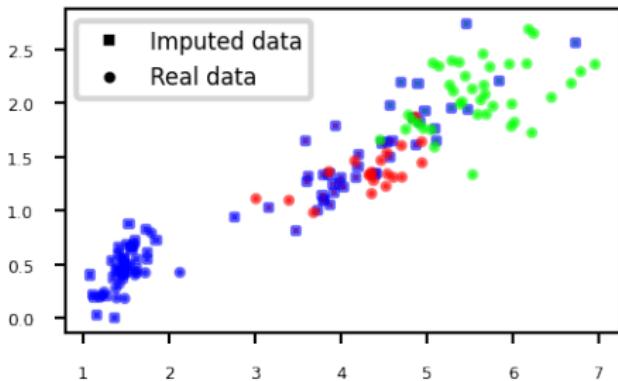


► Basic idea: low-rank approximation

- Replace missing values by 0
- Factorize \mathbf{X} with rank r : $\mathbf{X}^{n \times p} = \mathbf{U}^{n \times r} \mathbf{V}^{r \times p}$
 - With n data points and p features
 - Solved using gradient descent
- Recompute \mathbf{X} : now complete

Matrix Factorization (cont.)

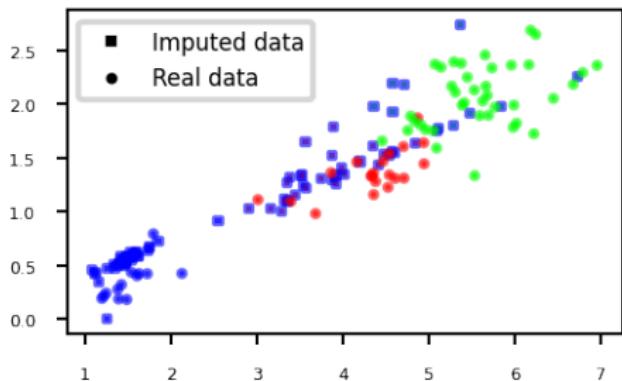
Matrix Factorization (ACC:0.973)



- ▶ Same basic idea, but smoother

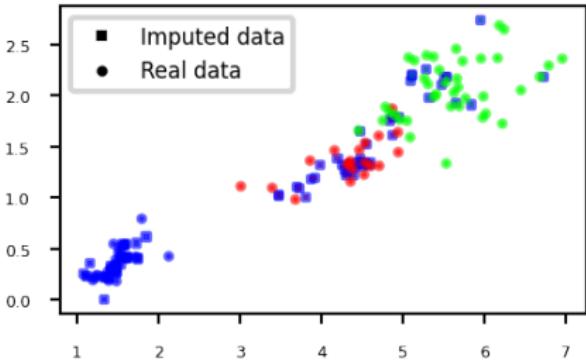
- Replace missing values by 0, compute SVD: $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^\top$
 - ▶ Solved with gradient descent
- Reduce eigenvalues by shrinkage factor: $\lambda_i = s \cdot \lambda_i$
- Recompute \mathbf{X} : now complete
- Repeat for m iterations

SoftImpute (ACC:0.967)



- ▶ Best method depends on the problem and dataset at hand. Use cross-validation.
- ▶ Iterative Imputation (MICE) generally works well for missing (completely) at random data
 - Can be slow if the prediction model is slow
- ▶ Low-rank approximation techniques scale well to large datasets

Iterative Imputation (RandomForest) (ACC:0.960)



► Problem:

- You have a majority class with many more examples than the minority class
- Or: classes are balanced, but associated costs are not (e.g., FN worse than FP)

► Solutions we've already covered:

- Add class weights to the loss function (give minority class more weight)
 - ▶ In practice: set `class_weight='balanced'`
- Change the prediction threshold to minimize false negatives or false positives

► Preprocessing strategies:

- Resample the data to correct the imbalance
 - ▶ Random or model-based
- Generate synthetic samples for the minority class

Handling imbalanced data (cont.)

- Build ensembles over different resampled datasets
- Combinations of these

- ▶ Copy the points from the minority class
- ▶ Randomly sample from the majority class (with or without replacement) until balanced
 - Optionally, sample until a certain imbalance ratio (e.g., 1/5) is reached
 - Multi-class: repeat with every other class
- ▶ Preferred for large datasets, often yields smaller/faster models with similar performance

Random Undersampling (cont.)

Original (AUC: 0.831) RandomUnderSampler (AUC: 0.830)



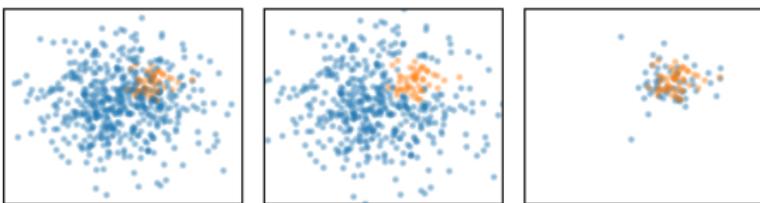
► Edited Nearest Neighbors

- Remove all majority samples that are misclassified by kNN (mode) or that have a neighbor from the other class (all).
- Remove their influence on the minority samples

► Condensed Nearest Neighbors

- Remove all majority samples that are *not* misclassified by kNN
- Focus on only the hard samples

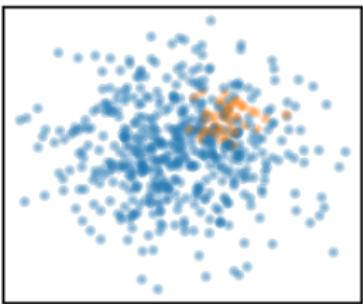
Original (AUC: 0.831) EditedNearestNeighbours (AUCCorRU) CondensedNearestNeighbour (AUC: 0.597)



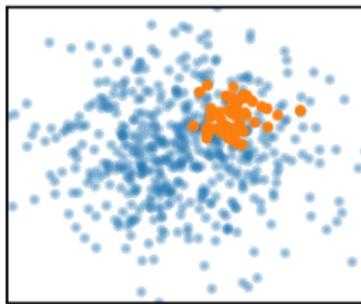
- ▶ Copy the points from the majority class
- ▶ Randomly sample from the minority class, with replacement, until balanced
 - Optionally, sample until a certain imbalance ratio (e.g. 1/5) is reached
- ▶ Makes models more expensive to train, doesn't always improve performance
- ▶ Similar to giving minority class(es) a higher weight (and more expensive)

Random Oversampling (cont.)

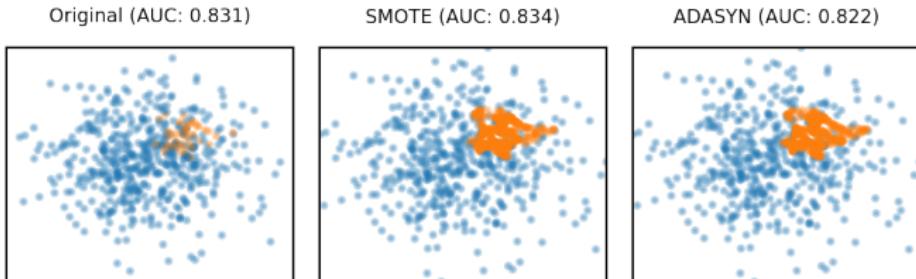
Original (AUC: 0.831)



RandomOverSampler (AUC: 0.829)



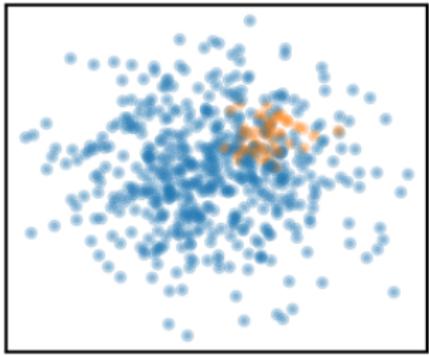
- ▶ Repeatedly choose a random minority point and a neighboring minority point
 - Pick a new, artificial point on the line between them (uniformly)
- ▶ May bias the data. Be careful never to create artificial points in the test set.
- ▶ ADASYN (Adaptive Synthetic)
 - Similar, but starts from 'hard' minority points (misclassified by kNN)



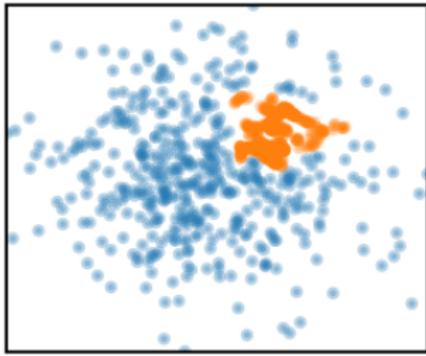
- ▶ Combines over- and under-sampling
- ▶ E.g. oversampling with SMOTE, undersampling with Edited Nearest Neighbors (ENN)
 - SMOTE can generate 'noisy' point, close to majority class points
 - ENN will remove up these majority points to 'clean up' the space

Combined techniques (cont.)

Original (AUC: 0.831)



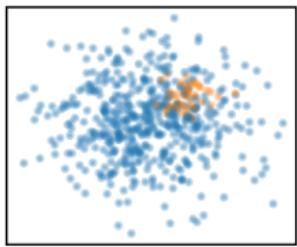
SMOTEENN (AUC: 0.878)



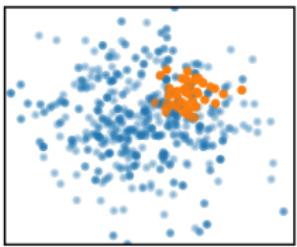
- ▶ Bagged ensemble of balanced base learners. Acts as a learner, not a preprocessor
- ▶ BalancedBagging: take bootstraps, randomly undersample each, train models (e.g. trees)
 - Benefits of random undersampling without throwing out so much data
- ▶ Easy Ensemble: take multiple random undersamplings directly, train models
 - Traditionally uses AdaBoost as base learner, but can be replaced

Ensemble Resampling (cont.)

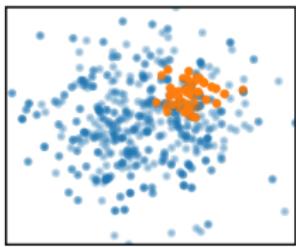
Original (AUC: 0.831)



EasyEnsembleClassifier (AUC: 0.841)



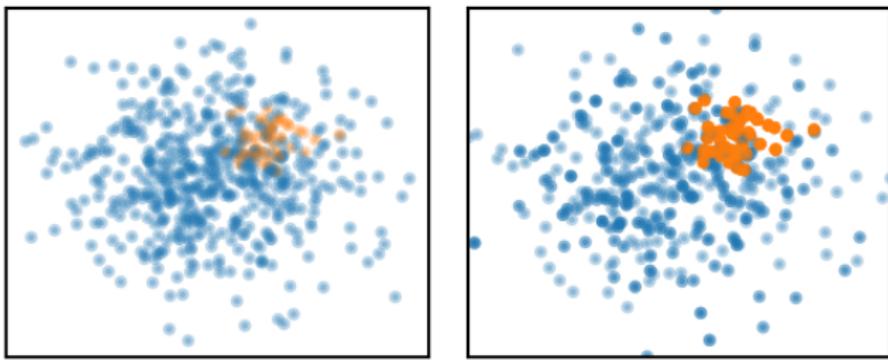
RandomizedBaggingClassifier (AUC: 0.851)



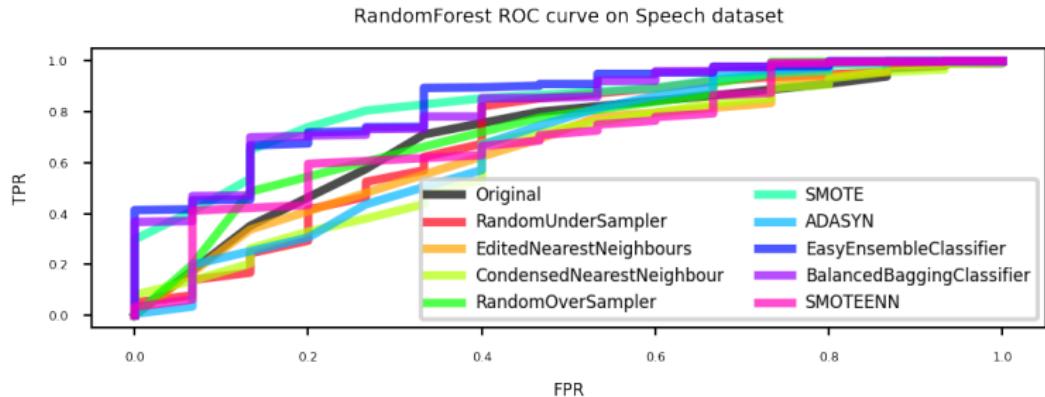
- ▶ The best method depends on the data (amount of data, imbalance,...)
 - For a very large dataset, random undersampling may be fine
- ▶ You still need to choose the appropriate learning algorithms
- ▶ Don't forget about class weighting and prediction thresholding
 - Some combinations are useful, e.g. SMOTE + class weighting + thresholding

Comparison (cont.)

Original (AUC: 0.831) EasyEnsembleClassifier (AUC: 0.840)



- ▶ The effect of sampling procedures can be unpredictable
- ▶ Best method can depend on the data and FP/FN trade-offs
- ▶ SMOTE and ensembling techniques often work well



- ▶ Data preprocessing is a crucial part of machine learning
 - Scaling is important for many distance-based methods (e.g. kNN, SVM, Neural Nets)
 - Categorical encoding is necessary for numeric methods (or implementations)
 - Selecting features can speed up models and reduce overfitting
 - Feature engineering is often useful for linear models
 - It is often better to impute missing data than to remove data
 - Imbalanced datasets require extra care to build useful models
- ▶ Pipelines allow us to encapsulate multiple steps in a convenient way
 - Avoids data leakage, crucial for proper evaluation
- ▶ Choose the right preprocessing steps and models in your pipeline
 - Cross-validation helps, but the search space is huge

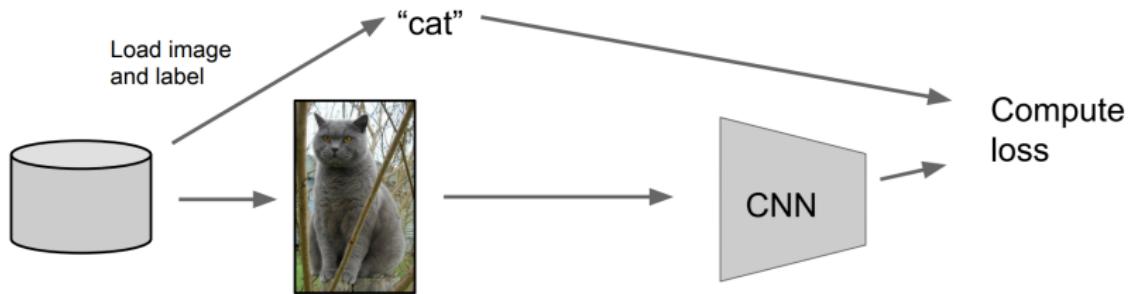
Summary (cont.)

- Smarter techniques exist to automate this process (AutoML)

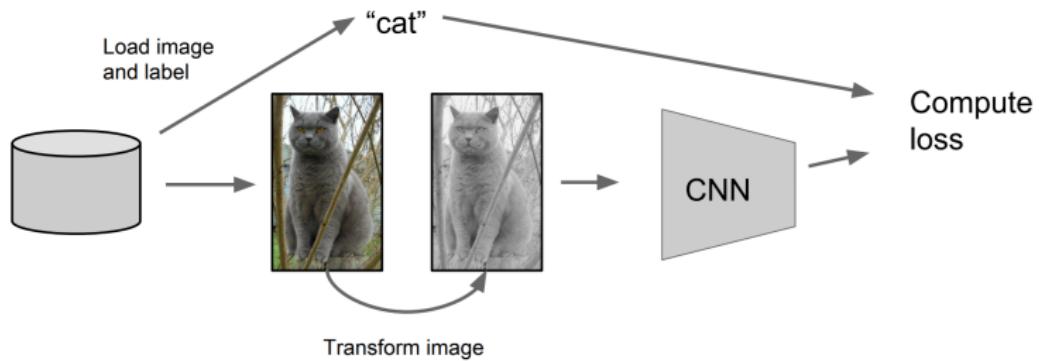
- [1] Open Machine Learning Course. *Open Machine Learning Course Repository*. GitHub 2024. <https://github.com/ml-course/master>.

- ▶ Technique to increase the diversity of training data without collecting new data
- ▶ Commonly used in image, text, and audio processing
- ▶ Helps reduce overfitting and improve generalization
- ▶ Especially useful for deep learning models

Data Augmentation

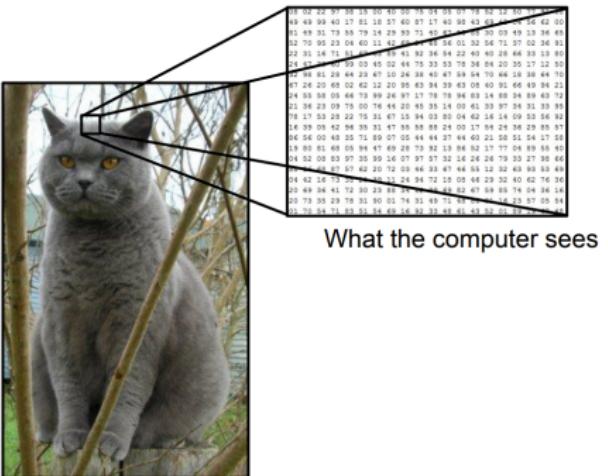


Data Augmentation



Data Augmentation

- ▶ Change the pixels without changing the label
- ▶ Train on transformed data
- ▶ **VERY** widely used



Data Augmentation

1. Horizontal flips



2. Random crops/scales

Training: sample random crops / scales

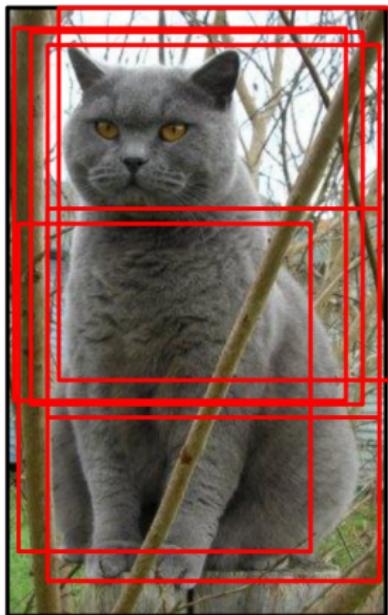
ResNet:

1. Pick random L in range [256, 480]
2. Resize training image, short side = L
3. Sample random 224×224 patch

Testing: average a fixed set of crops

ResNet:

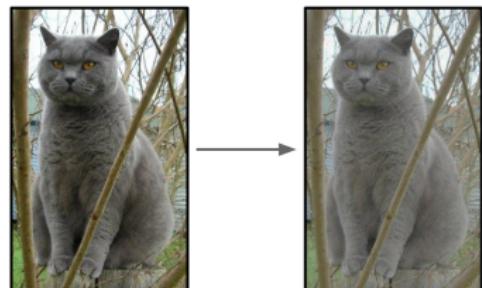
1. Resize image at 5 scales: {224, 256, 384, 480, 640}
2. For each size, use 10 224×224 crops:
4 corners + center, + flips



3. Color jitter

Simple:

Randomly jitter contrast



Complex:

1. Apply PCA to all [R, G, B] pixels in training set
2. Sample a “color offset” along principal component directions
3. Add offset to all pixels of a training image

(As seen in *[Krizhevsky et al. 2012]*, ResNet, etc)

4. Get creative!

Random mix/combinations of:

- ▶ translation
- ▶ rotation
- ▶ stretching
- ▶ shearing,
- ▶ lens distortions, . . . (go crazy)

Data Augmentation: Takeaway

- ▶ Simple to implement, use it
- ▶ Especially useful for small datasets
- ▶ Fits into framework of noise / marginalization

- [1] Stanford University. *CS231n: Convolutional Neural Networks for Visual Recognition*. 2024. <https://cs231n.stanford.edu/>.