Types of Machine Learning

(1) Supervised: $f(x_i)=y_i$ (most success)
e.g. spam detect, medical diag, click pred

(2) Unsupervised: x_i comes from $p(x)$ & we want to learn p (hard to evaluate)
e.g. clustering, dim reduc, topic modeling

(3) Reinforcement learning: neural nets, agents interact w/ environ & rewards
e.g. games, robotics, self-driving

(4) Other: semi-supervised, active learning, forecasting

Data Visualization

(1) Show data & maximize data-ink ratio
(2) Tools matter
(3) Labels – title, axes, color, legend, etc.

Channels for quantitative data
(1) Position
(2) Length
(3) Angle
(4) Area
(5) Volume
(6) Texture density
(7) Color saturation
(8) Color hue

* Think hard about aspect ratios, baselines, & 3D plotting

```
fig, ax = plt.subplots(n,m)
ax[0,0].plot(x,y)
ax[0,1].scatter(x,y, α=0.1)   # transparency value to help w/ overplotting
ax[1,0].hist(x, bins=1000)
ax[1,1].bar(range(len(y)), y)
```

Heatmap: color code to represent diff values
Hexgrid: density map (brighter colors = more pts)

Nearest Neighbors

* $k \to \infty$, overfitting!

Assign data label of avg of nearest neighbors

```
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
```

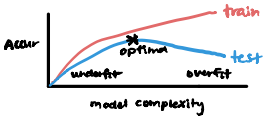
* fast to train
* large to store
* slow to predict

Regression

$\hat{y} = w^T x + b = \sum_i w_i x_i + b$

OLS: Minimize $\sum_i \| \hat{y}_i - y_i \|^2$ * unique soln if X full rank

Ridge: $\min_{w,b} \sum_i (\hat{y}_i - y_i)^2 + \alpha \|w\|_2^2$ always has soln if $\alpha \neq 0$



high α → underfit
small α → overfit

Lasso: $\min_{w,b} \sum_i (\hat{y}_i - y_i)^2 + \alpha \|w\|_1$, $\|w\|_1 = \sum_i |w_i|$

* sets some w_i to 0 = auto feature select
2 hyperparams

Elastic Net: $\min_{w,b} \sum_i (\hat{y}_i - y_i)^2 + \alpha_1 \|w\|_1 + \alpha_2 \|w\|_2^2$

Coeff of Determination:
$R^2(y,\hat{y}) = 1 - \frac{\sum_i (y_i - \hat{y})^2}{\sum_i (y_i - \bar{y})^2}$

* can be neg for biased data / test set

* For skewed data, log transforme better results

Classification

Binary case: $\hat{y} = \text{sign}(w^T x + b)$

Logistic Regression: * gives prob estimates

Loss → $\min_{w,b} \sum_i \mathbb{1}_{y_i \neq \hat{y}}$

$\log\left(\frac{p(y=1|x)}{p(y=-1|x)}\right) = w^T x + b \longrightarrow p(y=1|x) = \frac{1}{1+e^{-w^T x - b}}$

Loss function: $\sum_i \log(\exp(-y_i(w^T x_i + b)) + 1)$

Add a penalty: $\min_{w,b} C \sum_i \log(\exp(-y_i(w^T x_i + b)) + 1) + \|w\|$

C ∝ 1/α
big C less regular
small C more regular
$\|w\|_1$ or $\|w\|_2^2$

SVM: Draw boundary b/w classes w/ max margin (points exist outside margin, nearest pts = SV's)

Loss function: $\min_{w,b} C \sum_i \max(0, 1 - y_i(w^T x + b)) + \|w\|$

within margin → $y_i(w^T x_i + b) < 1$
smaller w → larger margin
* big C = fewer SV's

Linear SVM: soft margin, $\|w\|_1$ or $\|w\|_2^2$ hinge loss

Kernel SVM: $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ powerful, nonlinear models
long runtime for large data, scale!! good for sparse data

$\hat{y} = \text{sign}(\sum_i \alpha_i y_i (x_i^T x)) \longrightarrow \hat{y} = \text{sign}(\sum_i \alpha_i y_i k(x_i, x_j))$

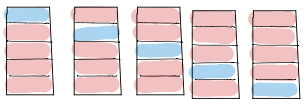
Multiclass Classification: 1 vs. rest, 1 vs. 1
* classify pt by all classifiers & return most common class
1 vs. rest: $\hat{y} = \text{argmax}_i\, w_i^T x + b_i$ — n classifiers
1 vs. 1: $\frac{n(n-1)}{2}$ classifiers, comp expensive

Softmax Loss: $P(y=i|x) = \frac{e^{w_i^T x + b_i}}{\sum_j e^{w_j^T x + b_j}}$

$\min_{w,b} -\sum_i \log p(y=y_i|x_i, w, b)$

Cross-Validation

k-fold, k=5 or 10 usually



* train 5 models & choose best params
* retrain on train set w/ these params

GridsearchCV

```
param_grid = {'n_neighbors': np.arange(1,15,2)}
grid = GridSearchCV(KNeighborsClassifier(), param_grid=param_grid, cv=10, return_train_score=True)
grid.fit(X_train, y_train)
grid.best_score_
grid.best_params_
grid.score(X_test, y_test)
```

Different Strategies

KFold: K models that split diff train & test
* BAD for ordered data → Shuffle!

Stratified KFold: Take some % of data for test & train across diff classes
* good for imbalanced classes

Leave-One-Out: KFold(n_folds = n_samples)
* high variance, takes too long

#2 ShuffleSplit: repeat sample test set w/ replace
#1 Repeated KFold: repeatedly shuffle & run KFold

GroupKFold: each group (i.e. city) should be entirely in train or test set

TimeSeriesSplit: "rolling window" of test/train (train should always be past of test)

Preprocessing / Pipelines

Scaling (esp impt for dist-based algorithms)

* StandardScaler: subtract mean to get var=1
MinMax Scaler: min val=0, max val=1
Robust Scaler: uses medians & quantiles
Normalizer: divide each row by Euclid norm

* For sparse data, scale but don't center

```
s = StandardScaler()
s.fit_transform(X_train, y_train)
s.transform(X_test, y_test)   # don't fit on test!
```

* MUST include preprocessing inside CV so we don't "leak info" and fit scaling on test data

```
pipe = make_pipeline(StandardScaler(), Ridge())
cross_val_score(pipe, X, y)
params_grid = {'ridge__alpha': [.01, .1 ... 1000]}
grid = GridSearchCV(pipe, param_grid=params_grid, n_jobs=-1, return_train_score=True)
grid.fit(X_train, y_train)
```

Missing Data (999, ?, np.NaN, "N/A", None ...)
* Sometimes missingness informative

Imputation: (1) mean/median (2) KNN *adaptive but slow (3) Regression (4) Matrix Fact *RF

```
imp = SimpleImputer(strategy="mean").fit(X_tr)
X_trans = imp.transform(X_tr)
knn_pipe = make_pipeline(KNNImputer(), StandardScaler(), LogisticRegression())
```

(3) Regression: train regression model to predict missing col
* Retrain after imputing missing data
* For categ data, add "cat_missing" category

Categorical Vars

Ordinal Encoding: Brooklyn → 1, Manhattan → 2, Queens → 3 } inherent order bad

last col = 1 - others
can create collinearity

OHE (drop 1 → which impt for L models)

```
pd.get_dummies(df, columns=['Boro'])
OneHotEncoder().fit(df) → transforms non-cat cols
```

Column Transformer

```
cat = df.dtypes == object
preprocess = make_column_transformer(
    (StandardScaler(), ~cat),
    (OneHotEncoder(), cat))
model = make_pipeline(preprocess, LogisticRegression())
```

Target / Impact Encoding

For high cardinality categories → instead of 70 OHE, one "response encoded" var (i.e. avg $ for zip)

Trees, Forests, Ensembles

Decision Tree: split based on threshold of feature w/ most info gain
can't extrapolate, fast though, no preprocess

Classification criteria: Gini $= \sum_k p_{mk}(1-p_{mk})$ (over all classes; prop of class k in node m)
Cross-Ent $= -\sum_k p_{mk} \log p_{mk}$

* Regression trees exist → $\bar{y}_m = \frac{1}{N_m} \sum_i y_i$ (MSE, MAE)

To prevent overfit: (1) prepruning (limit size while building)
(2) postpruning (cut branches after build)

Cost complexity: $K(T) + \alpha|T|$ → |T| = # leaf nodes; total leaf impurity
* BEST = greedy build tree then postprune w/ cost complexity

Probability = fraction class in leaf (without reg, no impurity)

Ensemble: Build many models & avg (more models better)

Bagging: build diff models by bootstrapping data (each tree usually uses 2/3 data) (picking random subset of features good too)

Extremely Random Tree: randomly choose thresh for feature, no bootstrap, fast!

Gradient Descent & Boosting

learning rate

To minimize $F(w)$, keep updating: $w_{i+1} \leftarrow w_i - \eta_i \frac{\partial F(w_i)}{\partial w}$

Stochastic GD: pick one x_i, compute partial gradient, & update w_i
* fast for large datasets

Boosting: "weak" learners ensemble to create "strong" one

$f_1(x) \approx y$
$f_2(x) \approx y - f_1(x)$
$f_3(x) \approx y - f_1(x) - f_2(x)$
$y = f_1(x) + f_2(x) + f_3(x) \ldots$

* Add in learning rate γ
$y = \gamma f_1(x) + \gamma f_2(x) \ldots$

Classification
$\min_y \sum_i \log(\exp(-y_i \hat{y}_i) + 1)$
Log Loss

Multiclass
$p(y=c|x) = \frac{e^{\hat{y}_c}}{\sum_j e^{\hat{y}_j}}$

* Too many trees = overfitting (stop adding trees when val acc stagnant)
(1) Pick # trees & learning rate
(2) Pick learning rate & use early stopping (max_depth good)

XGBoost speeds up tree building by binning & column subsampling

Model Evaluation

Binary Classification

Actual	pred −	pred +
−	True −	False +
+	False −	True +

$\text{Acc} = \frac{TP+TN}{TP+TN+FP+FN}$
* bad for class imbalance

Precision: $\frac{TP}{TP+FP}$ (+ pred value)
Recall: $\frac{TP}{TP+FN}$ (sensitivity, true + rate)

$F = \frac{2(\text{prec} \cdot \text{recall})}{\text{prec} + \text{recall}}$
* Generally care about TPR, FPR, precision

FPR: $\frac{FP}{FP+TN}$

Averaging:
Macro: $\frac{1}{|L|} \sum_{l \in L} R(y_l, \hat{y}_l)$ (recall) treats all classes =
Weighted: $\frac{1}{n} \sum_{l \in L} n_l R(y_l, \hat{y}_l)$ weights larger classes more
Balanced Acc: $\frac{1}{2}\left(\frac{TP}{TP+FP} + \frac{TN}{TN+FN}\right)$

y_pred = rf.predict_proba(X_test)[:,1] > 0.30

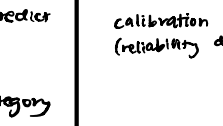
Prob-based
Precision-Recall curve: gives prec/recall over diff thresh
ROC curve: TPR vs. FPR * AUC = area under ROC (0.5 = random pred)
Avg precision: $\sum_k P(k) \cdot \Delta r(k)$ P(k) = prec & recall at k; sum over data pts k ranked by decision fn

Regression: R^2, MSE, MAE (R^2 & MSE sensitive to scaling & outliers)
MAPE: $\frac{100}{n} \sum_i \left|\frac{y_i - \hat{y}_i}{y_i}\right|$

Calibration & Imbalanced Data

Calibrate when prob more impt than label

Calibration plot (reliability diag)



* Fix by mapping classifier probs to better probs
$f_{\text{calib}}(s(x)) = p(y)$
model score
* use w/ val set, not train set
good for SVMs

Brier score (bin class) = $\frac{\sum_i (\hat{p}(y_i) - y_i)^2}{n}$

Platt scaling: $f_{\text{platt}} = \frac{1}{1+\exp(-w s(x) - b)}$

Isotonic Reg: flexible f_{calib} (learns increasing step fns)
* Optimum monotonic func on training data

Imbalanced Data (asymmetric cost or data)
Random undersampling, random oversampling
** RandomUnderSampler() RandomOverSampler() Slow
Instead of resampling, reweight loss func
$\min_{w,b} -C \sum_i c_{y_i} \log(\exp(-y_i(w^T x_i + b) + 1) + \|w\|_2^2$

*SMOTE good sometimes
Hgini = $\sum_k C_k p_{mk}(1-p_{mk})$
Ensemble resampling: random resampling for each tree

Linear model coeffs useless w/o scaling, w/ regularization, w/ feature correlation

Drop feature imptce: $I_i^{\text{drop}} = \text{Acc}(f, X, y) - \text{Acc}(f_{-i}, X_{-i}, y)$ diff models
Permutation imptce: $I_i^{\text{perm}} = \text{Acc}(f, X, y) - E_{\tilde{X}_i}[\text{Acc}(f(x_i, \tilde{X}_{-i}), y_{-i})]$ same model

LIME: sparse lin local model around each datapt
SHAP: dropout imptce for every subset of features
Partial Dep plots: $f_i^{\text{PDP}}(x_i) = E_{\tilde{X}_{-i}}[f(x_i, \tilde{X}_{-i})]$ marg dep of pred on 1 feature