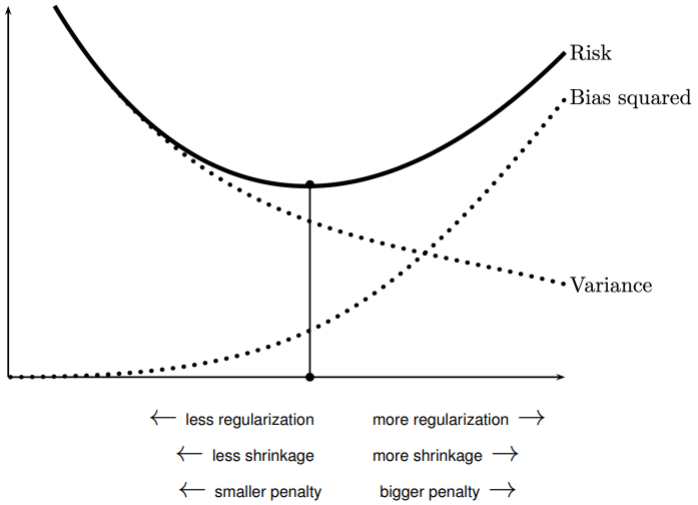
1 Bias-variance tradeoff

$$\begin{aligned}\mathrm{E}(\theta-\widehat{\theta})^2 &= \mathrm{E}(\theta-\mathbb{E}\widehat{\theta}+\mathbb{E}\widehat{\theta}-\widehat{\theta})^2 \\ &= \mathbb{E}(\theta-\mathbb{E}\widehat{\theta})^2 - 2\mathrm{E}\{(\theta-\mathbb{E}\widehat{\theta})(\widehat{\theta}-\mathbb{E}\widehat{\theta})\} + \mathbb{E}(\widehat{\theta}-\mathbb{E}\widehat{\theta})^2 \\ &= \mathbb{E}(\theta-\mathbb{E}\widehat{\theta})^2 - 2(\theta-\mathbb{E}\widehat{\theta})\mathrm{E}(\widehat{\theta}-\mathbb{E}\widehat{\theta}) + \mathbb{E}(\widehat{\theta}-\mathbb{E}\widehat{\theta})^2 \\ &= \mathbb{E}(\theta-\mathbb{E}\widehat{\theta})^2 + \mathbb{E}(\widehat{\theta}-\mathbb{E}\widehat{\theta})^2 \\ &= \mathrm{Bias}(\widehat{\theta})^2 + \mathrm{Variance}(\widehat{\theta})\end{aligned}$$



2 Cross Validation

	Iteration					
Obs	1	2	3	4	...	n
1	valid	train	train	train	...	train
2	train	valid	train	train	...	train
3	train	train	valid	train	...	train
4	train	train	train	valid	...	train
...	...	...	...	...	...	...
n	train	train	...	...	...	valid
MSE	MSE_1	MSE_2	MSE_3	MSE_4	...	MSE_n

LOOCV test error: $CV_{(n)} = \frac{1}{n}\sum_i MSE_i$

	Iteration						
Obs	1	2	3	4	...	*k*	
1	valid	train	train	train	...	train	fold 1
2	valid	train	train	train	...	train	
3	valid	train	train	train	...	train	
4	train	valid	train	train	...	train	
...	...	...	...	...	...	...	
$n-2$	train	train	...	...	...	valid	fold *k*
$n-1$	train	train	...	...	...	valid	
n	train	train	...	...	...	valid	
MSE	MSE_1	MSE_2	MSE_3	MSE_4	...	MSE_k	

K-Fold: $CV_{(k)} = \sum_b \frac{n_b}{n} MSE_b$ where n_b is the total observations in the b-th fold, and n is the total observations in the entire dataset. Suppose the fitted values can be written $\hat{Y} = HY$. The leave-one-out-cross-validation error is

$$R_{LOOCV} = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \widehat{Y}_{(-i)}\right)^2 = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{Y_i-\widehat{Y}_i}{1-H_{ii}}\right)^2$$

where H_{ii} is the i-th diagonal entry. This is the case for least squares and regularized multiple regression:
$H = X\left(X^TX\right)^{-1}X^T$ or $H = X\left(X^TX+\lambda I\right)^{-1}X^T$

3 Tree

Tree Build: (1) Cycle through predictors X_k for $k = 1,\ldots,p$. For each X_k:
- (Quantitative X_k) Consider cutpoints s (unique values of X_k) that divide up the region into two parts:

$$R_1(k,s) = \{i \mid X_{ik} < s\} \quad \text{and} \quad R_2(k,s) = \{i \mid X_{ik} \geq s\}$$

- Evaluate (for regression trees):

$$Q_k(s) = \sum_{i:i\in R_1(k,s)}\left(y_i - \bar{y}_{R_1}\right)^2 + \sum_{i:i\in R_2(k,s)}\left(y_i-\bar{y}_{R_2}\right)^2$$

Find the value of s that minimizes $Q_k(s)$. Call this s_k.
(2) Find the predictor X_k with the minimum $Q_1(s_1), Q_2(s_2), \ldots, Q_p(s_p)$. Make the first binary partition along predictor X_k at cut point s_k.
Cost-complexity pruning:

$$C(T) = \sum_{m=1}^{|T|}\sum_{i\in R_m}\left(y_i - \widehat{y}_{R_m}\right)^2 + \alpha|T|$$

$\alpha = 0$ implies the full tree, Larger α implies higher penalty for complexity of model. With increasing α:
(1) Grow a big tree on a training set. (2) Obtain a nested set of subtrees $T_L \subset \cdots \subset T_2 \subset T_1 \subset T$ corresponding to a sequence of α values. (3) Use K -fold cross-validation to identify the subtree that does best.
Classification Tree Impurity measures: Define node proportion of class k: $\widehat{p}_{mk} = \frac{1}{N_m}\sum_{x_i\in R_m} I(y_i = k)$ and $k(m) = \arg\max_k \widehat{p}_{mk}$
- Misclassification error: $1 - \widehat{p}_{mk(m)}$
- Gini index: $\sum_{k=1}^{K}\widehat{p}_{mk}\left(1-\widehat{p}_{mk}\right)$
- Entropy: $-\sum_{k=1}^{K}\widehat{p}_{mk}\log\widehat{p}_{mk}$

Bagging
Regression trees: Create B bootstrap samples, grow tree (without pruning) using each $\widehat{f}^{*1}, \widehat{f}^{*2}, \ldots \widehat{f}^{*B}$. For prediction at x , we take an average: $\widehat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\widehat{f}^{*b}(x)$
Classification trees: $\widehat{f}_{\text{bag}}(x)$ is decided by majority vote.
OOB Estimation: For each bagged tree, we can make predictions for the OOB observations. At the end, we can aggregate over all predictions for the i-th observation to arrive at a OOB prediction $\widehat{y}_i$. We can compute prediction error based on these OOB predictions $\widehat{y}_1, \ldots, \widehat{y}_n$.
Random Forests:
(1) For b=1 to B:
(a) Draw a bootstrap sample Z^* of size n from the training data
(b) Grow a random-forest tree T_b to the bootstrapped data, recursively repeating following steps, until minimum node size reached: i. Select m variables at random from the p variables ii. Pick the best variable/split-point among the m iii. Split the node into two children nodes

(2) Output the ensemble fo trees $\{T_b\}_{b=1}^B$. To make a prediction at a new point x :

Regression: Average $\hat{f}_{rf}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$

Classification: Majority vote of the individual trees

4 PCA

Algorithm (1) Center the data: $x_i \mapsto x_i - \frac{1}{n}\sum_{j=1}^{n} x_j = x_i - \bar{x}$ (2) Compute the $d \times d$ sample covariance $S = \frac{1}{n}\sum_{i=1}^{n} x_i x_i^T$. Note that $\frac{1}{n}\sum_i (x_{ij} - \bar{x})^2$ is the sample variance of j th coordinate of data.(3) Find the first k eigenvectors of S ,$\phi_1, \ldots, \phi_k \in \mathbb{R}^d$, $S\phi_j = \lambda_j \phi_j$(4) Project the data onto those k vectors: $x_i \mapsto \bar{x} + \left(\phi_1^T x_i\right)\phi_1 + \ldots + \left(\phi_k^T x_i\right)\phi_k$

PCA is an unsupervised method

- Finds directions of greatest variation in the data
- The directions are called the principal vectors; the weightings on the vectors are called the principal components
- The first few vectors may be interpretable
- Orthogonality makes interpretation difficult for the higher components
- Can be used for visualization or dimensionality reduction

- Prediction: accurately predict Y for new observations
  Inference: explain the underlying relationship between Y and X
- regression
  - linear regression: Fitting a straight line through the data.
  - knn: k-nearest neighbors regression. Average together the yi for xi close to x
  - mse/rss

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2$$

  - overfitting: A method is overfitting the data when it has a small training MSE but a large test MSE.
  - The **least squares** approach selects coefficients β0 and β1 that minimize the residual sum of squares

$$\hat{\beta} = (X^T X)^{-1} X^T y.$$

$$\hat{y}_i = \hat{a} + \hat{b}x_i$$

$$\bar{y} = \hat{a} + \hat{b}\bar{x}$$

$$\hat{b} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Least squares coefficients correspond to minimum of a quadratic surface

  - R^2(multiple R-squared):the proportion of variability in y explained by the model

$$\underbrace{\sum(y_i - \bar{y})^2}_{\text{total sum of squares}(TSS)} = \underbrace{\sum(\hat{y}_i - \bar{y})^2}_{\text{explained sum of squares}(ESS)} + \underbrace{\sum(y_i - \hat{y}_i)^2}_{\text{residual sum of squares}(RSS)}$$

$$R^2 = \frac{ESS}{TSS} = 1 - \frac{RSS}{TSS}$$

- bayes decision rule

Binary classifier *h*: function from $\mathcal{X}$ to $\{0, 1\}$.

Linear if exists a function $H(x) = \beta_0 + \beta^T x$ such that $h(x) = 1$ if $H(x) > 0$; 0 otherwise.

$H(x)$ also called a *linear discriminant function*. Decision boundary: set $\{x \in \mathbb{R}^d : H(x) = 0\}$

Classification risk, or *error rate*, of *h*:

$$R(h) = \mathbb{P}(Y \neq h(X))$$

and the *empirical classification error* or *training error* is

$$\hat{R}(h) = \frac{1}{n}\sum_{i=1}^{n} I(h(x_i) \neq y_i).$$

Theorem. The rule *h* that minimizes $R(h)$ is

$$h^*(x) = \begin{cases} 1 & \text{if } m(x) > \frac{1}{2} \\ 0 & \text{otherwise} \end{cases}$$

where $m(x) = \mathbb{E}(Y \mid X = x) = \mathbb{P}(Y = 1 \mid X = x)$ denotes the regression function.

The rule h^* is called the *Bayes rule*. The risk $R^* = R(h^*)$ of the Bayes rule is called the *Bayes risk*. The set $\{x \in \mathcal{X} : m(x) = 1/2\}$ is called the *Bayes decision boundary*.

From Bayes' theorem

$$\mathbb{P}(Y = 1 \mid X = x) = \frac{p(x \mid Y = 1)\mathbb{P}(Y = 1)}{p(x \mid Y = 1)\mathbb{P}(Y = 1) + p(x \mid Y = 0)\mathbb{P}(Y = 0)} = \frac{\pi_1 p_1(x)}{\pi_1 p_1(x) + (1 - \pi_1)p_0(x)}.$$

where $\pi_1 = \mathbb{P}(Y = 1)$. So,

$$m(x) > \frac{1}{2} \quad \text{is equivalent to} \quad \frac{p_1(x)}{p_0(x)} > \frac{1 - \pi_1}{\pi_1}.$$

Thus the Bayes rule can be rewritten as

$$h^*(x) = \begin{cases} 1 & \text{if } \frac{p_1(x)}{p_0(x)} > \frac{1 - \pi_1}{\pi_1} \\ 0 & \text{otherwise.} \end{cases}$$

- logistic regression

$$logit(\hat{p}(x)) = \hat{\beta}_0 + \hat{\beta}_1 x$$

decision boundary is given by $\{x : x^T\hat{\beta} = 0\}$.

- fitting LR

$$L_i(\beta) = p_i^{y_i} \cdot (1 - p_i)^{1 - y_i} = \left(\frac{e^{x_i^T\beta}}{1 + e^{x_i^T\beta}}\right)^{y_i} \cdot \left(1 - \frac{e^{x_i^T\beta}}{1 + e^{x_i^T\beta}}\right)^{1 - y_i}$$

$$\ell_i(\beta) = y_i \log\left(\frac{e^{x_i^T\beta}}{1 + e^{x_i^T\beta}}\right) + (1 - y_i)\log\left(\frac{1}{1 + e^{x_i^T\beta}}\right) = y_i x_i^T\beta - \log(1 + e^{x_i^T\beta})$$

$$\ell(\beta) = \sum\left(y_i x_i^T\beta - \log(1 + e^{x_i^T\beta})\right)$$

$$\hat{\beta} = \arg\min_{\beta}(y - \beta)^2 + \lambda\beta^2$$

Solution: $\hat{\beta} = \frac{y}{1+\lambda}$. As λ gets large, $\hat{\beta}$ shrinks to zero.

- regualrization
- 2 classifiers
  - Generative models model both the input X and the output Y. estimate the joint distribution by maximizing the joint likelihood: p(x,y). assume p(x|y)

$$X \mid y = 0 \sim N(-1, \sigma^2) \quad X \mid y = 1 \sim N(1, \sigma^2) \quad X \mid y = 2 \sim N(2, \sigma^2)$$

  - Discriminative models model only the output Y given X. estimate by maximizing the conditional likelihood.
- Stochastic Gradient Descent
  - steps
    1. Read data item x
    2. Make a prediction $\hat{y}(x)$
    3. Observe the true response/label y
    4. Update the parameters β so $\hat{y}$ is closer to y
  - idea
    - For each parameter β_j, see what happens to the loss if that parameter is increased a little bit.
    - If the loss goes down (up), then increase (decrease) β_j proportionately
    - Do this simultaneously for all of the parameters
    - Rinse and repeat
  - general beta and loss update

$$\beta_j \longleftarrow \beta_j - \eta\frac{\partial L(y, \beta^T x)}{\partial \beta_j}$$

$$\beta \longleftarrow \beta - \eta\nabla_\beta L(y, \beta^T x) \quad \text{(vector notation)}$$

$$L(\beta + \eta v) \approx L(\beta) + \eta v^T \nabla L(\beta)$$

$$L(\beta - \eta\nabla L(\beta)) \approx L(\beta) - \eta\|\nabla L(\beta)\|^2$$

- sgd for linear regression

Change β_j by a little bit:

$$\beta_j \to \beta_j + \varepsilon$$

What happens to the squared error?

$$(y - \hat{y})^2 \to (y - \hat{y} - \varepsilon x_j)^2$$

$$\approx (y - \hat{y})^2 + \underbrace{-2(y - \hat{y})x_j}_{\text{derivative of loss}}\varepsilon$$

Use adjustment

$$\beta_j \to \beta_j \overbrace{-\eta \cdot \text{derivative of loss}}^{\varepsilon}$$

$$= \beta_j + \eta \cdot 2(y - \hat{y})x_j$$

Squared error then decreases:

$$(y - \hat{y})^2 \approx (y - \hat{y})^2 - 4\eta(y - \hat{y})^2 x_j^2$$

```
dbeta = X.T @ (X@beta + intercept - y)  # line (
dintercept = np.sum(X@beta + intercept - y)  # l

beta = beta - step_size * dbeta
intercept = intercept - step_size * dintercept
```

- sgd for logistic regression

$$\beta_j \longleftarrow \beta_j + \eta(y - p(x))x_j$$

$$\beta_j x_j \longleftarrow \beta_j x_j + \eta(y - p(x))x_j^2$$

$$p(x) = \frac{1}{1 + \exp(-\beta^T x)}$$

- lr. 1/t : decreasing lr

Linear Regression

```
X = sm.add_constant(x_week)
model = sm.OLS(y_week, X)
result = model.fit()
beta = [result.params[0], result.params[1]]
```

```
# form an array with 1, x, x^2

x_week = x[is_weekday]
y_week = y[is_weekday]
X = np.array([np.ones(len(x_week)), x_week, x_week**2])
X = X.T

# fit a linear model
model = sm.OLS(y_week, X)
result = model.fit()
beta = [result.params[0], result.params[1], result.params[2]]

# plot the result
week_dates = dates[is_weekday]
plt.scatter(dates, y, alpha=.5)
plt.plot(x, beta[0] + beta[1]*x + beta[2]*x**2, color='red', linewidth=2)
_ = plt.xticks(dates[[0, 40, 80, 120, len(dates)-1]], rotation=90)
```

Logistic Regression

```
X = iris.data
y = iris.target

lr = LogisticRegression(penalty='l2', C=.1, multi_class='multinomial')
lr_error_rate = []
trials = 100
train_percent = np.linspace(.1,.9,num=9)

# your code here
from tqdm import tqdm
for p in tqdm(train_percent):
    errs = []
    for trial in np.arange(trials):
        X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=p)
        lr.fit(X_train, y_train)
        err = np.mean(lr.predict(X_test) != y_test)
        errs.append(err)
    this_err = np.mean(errs)
    lr_error_rate.append(this_err)

plt.plot(train_percent, lr_error_rate)
plt.xlabel('train percent')
_ = plt.ylabel('error')
```

```
beta = 0*param
steps = 100000

delta = .1
for t in tqdm(range(1,steps)):
    i = np.random.choice(n)
    x_i = X_train[i]
    y_i = y_train[i]
    p = 1 / (1 + np.exp(-np.dot(x_i,beta)))
    beta = beta + delta/np.sqrt(t) * (y_i - p) * x_i
```

```
for i in np.arange(len(bandwidths)):
    sq_bias[i] = np.mean((np.mean(fhat[i], axis=0) - f)**2)
    variance[i] = np.mean(np.var(fhat[i], axis=0))
```

```
from sklearn import tree
dtree = tree.DecisionTreeClassifier(min_samples_leaf=75)
dtree = dtree.fit(X, y)
fig = plt.figure(figsize=(20,20))
_ = tree.plot_tree(dtree, filled=True, feature_names = X.columns)
oob_error = []
num_trees = np.arange(50, 180, 20)
rf = ensemble.RandomForestRegressor(min_samples_leaf=100, max_features=4, \
                                    criterion='mse', oob_score=True)

for m in tqdm(num_trees):
    rf.set_params(n_estimators=m)
    model = rf.fit(X, y)
    oob_error.append(1-model.oob_score_)
```

```
images = X[(y==3), :]
avgimg = images.mean(0)
_ = plt.imshow(avgimg.reshape((28, 28)), cmap=plt.cm.gray.reversed())
```

```
cimages = np.subtract(images, avgimg)
_ = plt.imshow(cimages[7].reshape((28, 28)), cmap=plt.cm.gray.reversed())
plt.show()
_ = plt.imshow(np.add(cimages[7], avgimg).reshape((28, 28)), cmap=plt.cm.gray.reversed())
```

+ Code + Text

```
height = cimages.shape[1]
width = cimages.shape[2]
cimages = cimages.reshape(cimages.shape[0], height*width)
cimages.shape
```

```
num_components = 25
pca = PCA(num_components).fit(cimages)
principal_vectors = pca.components_
principal_vectors = principal_vectors.reshape((num_components, height, width))
pcs = pca.fit_transform(cimages)
capprox = pca.inverse_transform(pcs)
labels = ['principal vector %d' % (i+1) for i in np.arange(num_components)]
plot_images(principal_vectors, labels, height, width, int(num_components/5.), 5)
ratio = pca.explained_variance_ratio_.sum()
print('Variance explained by first %d principal vectors: %.2f%%' % (num_components, ratio*100))
```

```
pcs.shape
cimages[0].shape
v = principal_vectors[0].reshape(784,1)
np.dot(cimages[0], v)
```