

# Training Models

Elliott Ash

Text Data Course, Bocconi 2018

# Different Methods for Different Goals

- ▶ **Supervised: Pursuing a known goal – prediction or classification.**
- ▶ Unsupervised: Unknown goal, let the computer summarize the data.

# Approximating $Y = f(X)$

- ▶ We want to predict a real-valued outcome  $Y$  given  $X$ , that is, constructing an approximation of the function  $f(X)$ .
  - ▶ With high-dimensionality and multi-collinearity, normal regression methods do not work.
- ▶ Supervised learning:
  - ▶ regularized regression
  - ▶ random forests
  - ▶ cross-validation

# OLS Regression

- ▶ Consider the linear model

$$Y_i = X_i' \beta + \epsilon_i$$

where  $Y_i$  and all elements of  $X_i$  have been de-meanned and standardized to  $s.d. = 1$ .

- ▶ OLS assumptions:
  - ▶  $X_i$  uncorrelated with  $\epsilon_i$ 
    - ▶ Let's just assume this for now; will come back later.
  - ▶ Columns of  $X_i$  are not highly collinear.
    - ▶ In the case of word/n-gram frequency data, this is not a good assumption.

# Univariate OLS Regressions to Rank Predictive Features

- ▶ Consider the univariate regression

$$Y_i = \beta_w x_i^w + \epsilon_i$$

for each text feature  $w$  (e.g., relative word or n-gram frequency).

- ▶ Can be estimated with OLS.
  - ▶ Can add fixed effects, or even better: residualize  $Y$  and  $X$  on fixed effects before running any regressions.
  - ▶ Robust or clustered standard errors is optional, if the goal is just to rank predictors or filter out noise features.

# OLS Regression in Python statsmodels

- ▶ One could write a DO file to run these regressions in Stata.
  - ▶ But the loops and data saving would be tricky with so many feature variables.

```
import statsmodels.formula.api as smf
```

```
tstats, betas = [], []
```

```
for xvar in vocab: # loop through the words in v
```

```
    print(xvar)
```

```
    model = smf.ols('Y_~_ %s' % xvar, data=df4)
```

```
    result = model.fit()
```

```
    tstats.append(result.tvalues[1])
```

```
    betas.append(result.params[1])
```

```
stats = list(zip(vocab, tstats))
```

```
stats.sort(key = lambda x: x[1], reverse=True)
```

```
stats[:10]
```

```
stats[-10:]
```

# Ash-Morelli-Osnabrugge: Interpreting Classes

- ▶ Recall the policy topic model, which we applied to New Zealand parliamentary speeches:
  - ▶ We then looked at the n-grams in that corpus that were predictive of the topic classification.
  - ▶ Formally, for each topic  $k$  and each phrase  $m$ , regress

$$\Pr(y_i = k) = \alpha + \delta_m x_i^m + \epsilon_i$$

where  $\Pr(y_i = k)$  is the predicted probability that speech  $i$  is topic  $k$ , and  $x_i^m$  is the tf-idf frequency of phrase  $m$  in speech  $i$ .

- ▶ select phrases with highest positive t-statistic for  $\delta_m$

# Agriculture and Education Topics





# Estimating OLS

- ▶ OLS minimizes the mean squared error:

$$\min_{\hat{\theta}} \frac{1}{m} \sum_{i=1}^m (\mathbf{x}_i' \hat{\theta} - y_i)^2$$

where  $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{ip})$

- ▶ This has a closed form solution

$$\hat{\theta} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y}$$

- ▶ Most machine learning models do not have a closed form solution.
  - ▶ objective must be minimized using some other optimization algorithm.

# Gradient Descent

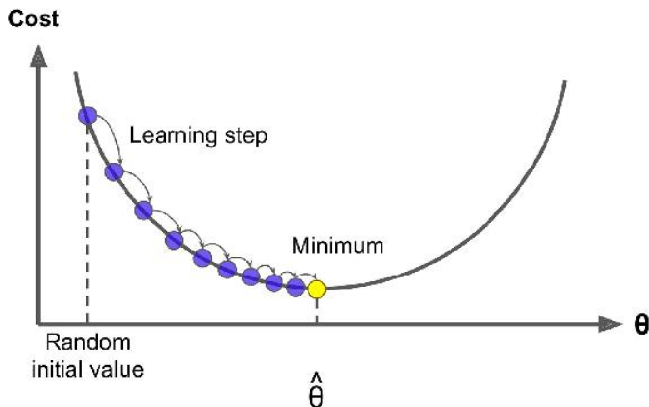
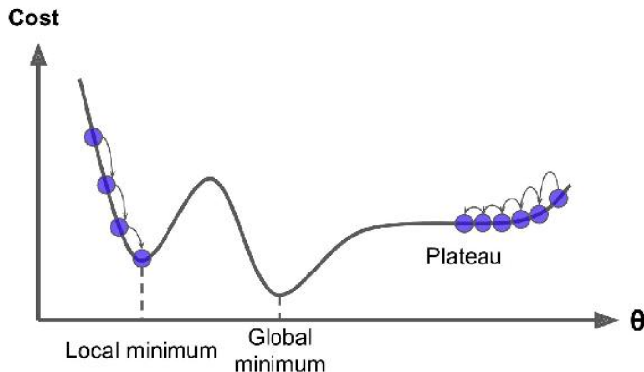


Figure 4-3. Gradient Descent

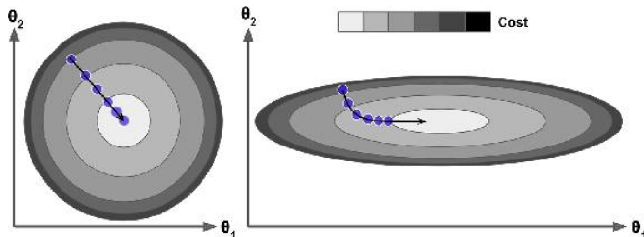
- ▶ Gradient descent measures the local gradient of the error function, and then steps in that direction.
  - ▶ Once the gradient equals zero, you have reached a minimum.

# Gradient Descent Pitfalls



- Gradient descent will find a local minimum, not necessarily a global minimum. And it can get stuck on plateaus.

# Gradient Descent and Scaling



- ▶ When using gradient descent, all features should be standardized to the same scale to speed up convergence.

# Gradient Descent Implementation

- ▶ The partial derivative of MSE for feature  $j$  is

$$\frac{\partial \text{MSE}}{\partial \theta_j} = \frac{2}{m} \sum_{i=1}^n (\mathbf{x}'_i \hat{\theta} - y_i) x_{ij}$$

- ▶ The gradient is the vector of these partial derivatives is

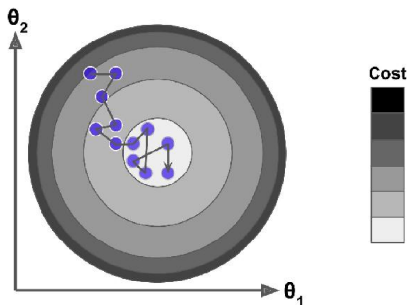
$$\nabla_{\theta} \text{MSE} = \begin{bmatrix} \frac{\partial \text{MSE}}{\partial \theta_0} \\ \frac{\partial \text{MSE}}{\partial \theta_1} \\ \vdots \\ \frac{\partial \text{MSE}}{\partial \theta_j} \end{bmatrix} = \frac{2}{m} \mathbf{X}' (\mathbf{X}' \theta - \mathbf{y})$$

- ▶ Gradient descent step:

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta} \text{MSE}$$

# Stochastic Gradient Descent

- SGD picks a random instance in the training set and computes the gradient only for that single instance.

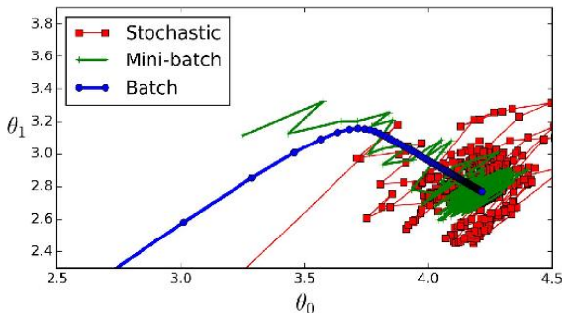


```
from sklearn.linear_model import SGDClassifier  
sgd_clf = SGDClassifier(n_iter=50, penalty=None, eta0=.1)  
sgd_clf.fit(X, y)
```

- Much faster to train, but can still bounce around even after it is close to the minimum.

# Mini-Batch Gradient Descent

- ▶ A compromise between gradient descent and stochastic gradient descent is mini-batch gradient descent, which selects a small number of rows (a “mini-batch”) for gradient compute, rather than a single row.



# The Problem of Overfitting

```
m = 100
X = 6 * np.random.rand(m,1) - 3
y = 0.5 * X ** 2 + X + 2 + np.random.randn(m,1)

from sklearn.preprocessing import PolynomialFeatures
poly_2 = PolynomialFeatures(degree=2) # also adds interactions
X_poly_2 = poly_2.fit_transform(X)

poly_300 = PolynomialFeatures(degree=300)
X_poly_300 = poly_300.fit_transform(X)

lin_reg = LinearRegression()
scores_1 = cross_val_score(lin_reg, X, y, cv=3, n_jobs=3)
scores_2 = cross_val_score(lin_reg, X_poly_2, y, cv=3, n_jobs=3)
scores_3 = cross_val_score(lin_reg, X_poly_300, y, cv=3, n_jobs=3)

print(scores_1.mean())
print(scores_2.mean())
print(scores_3.mean())
```



# The Problem of Overfitting

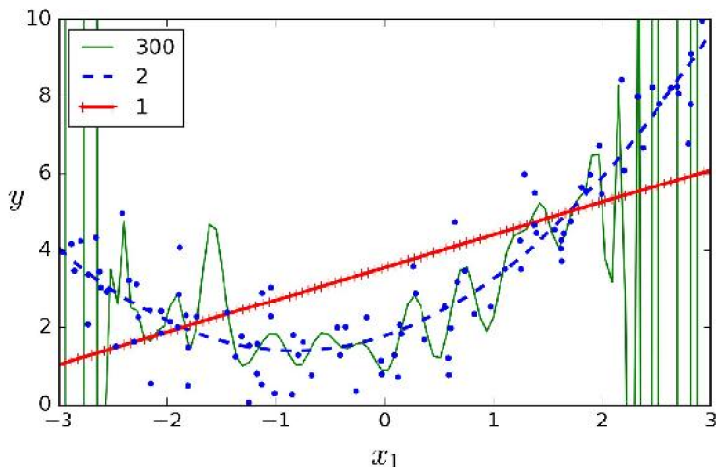


Figure 4-14. High-degree Polynomial Regression

# Bias-Variance Tradeoff

- ▶ *Bias*

- ▶ Error due to wrong assumptions, such as summing data is linear when it is quadratic.
- ▶ Will underfit the training data

- ▶ *Variance*

- ▶ Error due to excess sensitivity to small variations in the training data.
- ▶ A model with high variance is likely to overfit the training data.

- ▶ *Irreducible error*

- ▶ Error due to noise in the data.

# Training Notes

- ▶ In general, increasing a model's complexity will increase variance and reduce bias.
- ▶ If the model is underfitting:
  - ▶ adding more training data will not help.
  - ▶ use a more complex model
- ▶ If the model is overfitting:
  - ▶ adding more training data may help
  - ▶ or use regularization

# Lasso, Ridge, and Elastic Net

- ▶ Lasso and ridge regression are tools for dealing with large feature sets where:
  - ▶ models have multicollinearity that causes bias
  - ▶ models tend to overfit
  - ▶ models are computationally costly to fit
- ▶ These algorithms work by constraining estimated parameter sizes.

# Lasso Regression

- ▶ The Lasso cost function is

$$J(\theta) = \text{MSE}(\theta) + \alpha_1 \sum_{i=1}^n |\theta_i|$$

- ▶  $i$  indexes over  $n$  features
  - ▶  $\alpha_1$  is a hyperparameter setting the strength of the L1 penalty
- ▶ Lasso automatically performs feature selection and outputs a sparse model.

```
from sklearn.linear_model import Lasso  
lasso_reg = Lasso(alpha=0.1)  
lasso_reg.fit(X,y)
```

# Ridge Regression

- ▶ The Ridge cost function is

$$J(\theta) = \text{MSE}(\theta) + \alpha_2 \frac{1}{2} \sum_{i=1}^n \theta_i^2$$

- ▶  $i$  indexes over  $n$  features
  - ▶  $\alpha_2$  is a hyperparameter setting the strength of the L2 penalty
- ▶ It turns out that the Ridge estimator, like OLS, has a closed-form solution:

$$\hat{\theta}_{\text{Ridge}} = (X'X + \alpha_2 \mathbf{I}_n)^{-1} X' \mathbf{y}$$

where  $\mathbf{I}_n$  is the identity matrix.

- ▶ But it can also be solved by (stochastic) gradient descent.

```
from sklearn.linear_model import Ridge
ridge_reg = Ridge(alpha=1)
ridge_reg.fit(X,y)
sgd_ridge_reg = SGDRegressor(penalty="l2")
sgd_ridge_reg.fit(X,y)
```

# Elastic Net

- ▶ Elastic Net uses both L1 and L2 penalties:

$$J(\theta) = \text{MSE}(\theta) + \lambda\alpha_1 \sum_{i=1}^n |\theta_i| + (1 - \lambda)\alpha_2 \frac{1}{2} \sum_{i=1}^n \theta_i^2$$

- ▶ in general, elastic net is preferred to lasso, which can behave erratically when the number of features is greater than the number of rows, or when some features are highly collinear.
- ▶ For elastic net, as with Lasso and Ridge, the hyperparameters can be selected by cross-validation.

```
from sklearn.linear_model import ElasticNetCV
enet_reg = ElasticNetCV(alphas=[.01,.1,1],
                        l1_ratio=[.01,.1,.5,.9,.99,1])
enet_reg.fit(X,y)
```

## Scaling while maintaining sparsity

- ▶ Regularization penalties are designed to work with scaled data.
  - ▶ An important feature of text data is sparsity, which is lost when taking out the mean:

$$\tilde{x}_i = \frac{x_i - \bar{\mathbf{x}}}{SD[\mathbf{x}]}$$

- ▶ Solution:

$$\tilde{x}_i = \frac{x_i}{SD[\mathbf{x}]}$$

```
from sklearn.preprocessing import StandardScaler  
sparse_scaler = StandardScaler(with_mean=False)  
X_sparse = sparse_scaler.fit_transform(X)
```



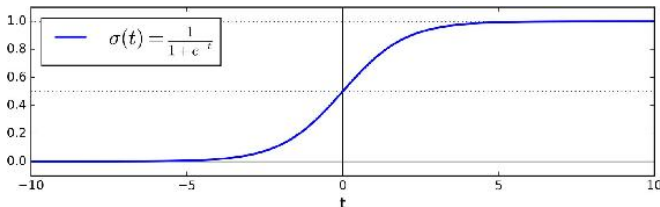
# Logistic Regression

- ▶ Like OLS, logistic regression computes a weighted sum of the input features to predict the output.
  - ▶ But rather than output the sum directly, it transforms the sum using the logist function.

$$\hat{p} = \Pr(Y_i = 1) = \sigma(\theta' \mathbf{x})$$

where  $\sigma(\cdot)$  is the sigmoid function

$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$



- ▶ For the binary version of the classifier, the prediction  $\hat{Y} \in \{0,1\}$  is determined by whether  $\hat{p} \geq .5$ .

# Logistic Regression Cost Function

- ▶ The cost function to minimize is the

$$J(\theta) = \underbrace{-\frac{1}{m}}_{\text{negative}} \sum_{i=1}^m \left[ \underbrace{y_i}_{y_i=1} \underbrace{\log(\hat{p}_i)}_{\log \text{ prob } y_i=1} + \underbrace{(1-y_i)}_{y_i=0} \underbrace{\log(1-\hat{p}_i)}_{\log \text{ prob } y_i=0} \right]$$

- ▶ this does not have a closed form solution
- ▶ but it is convex, so gradient descent will find the global minimum.
- ▶ Just like linear models, logistic can be regulated with L1 or L2 penalties, e.g.:

$$J(\theta) = \log \text{ loss} + \alpha_2 \frac{1}{2} \sum_{i=1}^n \theta_i^2$$

# Multinomial Logistic Regression

- ▶ Logistic can be generalized to multiple classes.
  - ▶ When given an instance  $\mathbf{x}_i$ , multinomial logistic computes a score  $s_k(\mathbf{x}_i)$  for each class  $k$ ,

$$s_k(\mathbf{x}_i) = \theta'_k \mathbf{x}_i$$

- ▶ If there are  $n$  features and  $K$  output classes, there is a  $K \times n$  parameter matrix  $\Theta$ , where the parameters for each class are stored as rows.
- ▶ Using the scores, probabilities for each class are computed using the softmax function

$$\hat{p}_k(\mathbf{x}_i) = \Pr(Y_i = k) = \frac{\exp(s_k(\mathbf{x}_i))}{\sum_{j=1}^K \exp(s_j(\mathbf{x}_i))} = \frac{e^{\theta_k \mathbf{x}_i}}{\sum_{j=1}^K e^{\theta_j \mathbf{x}_i}}$$

- ▶ And the prediction  $Y_i \in \{1, \dots, K\}$  is determined by the highest-probability category.

# Multinomial Logistic Cost Function

- ▶ The binary cost function generalizes to the cross entropy

$$J(\theta) = \underbrace{-\frac{1}{m}}_{\text{negative}} \sum_{i=1}^m \sum_{k=1}^K \underbrace{\mathbf{1}[y_i = k]}_{y_i=k} \underbrace{\log(\hat{p}_k(\mathbf{x}_i))}_{\log \text{ prob}_{y_i=k}}$$

- ▶ again, this is convex, so gradient descent will find the global minimum.

```
from sklearn.linear_model import LogisticRegression
logistic = LogisticRegression(C=1, # default L2 penalty
                              class_weight='balanced')
```

```
scores = cross_val_score(logistic ,
                          X,
                          y,
                          cv=3,
                          n_jobs=3)
```

```
scores.mean() , scores.std()
```

# Comparative Manifesto Project Corpus

- ▶ 44,020 annotated English-language political statements
  - ▶ hundreds of political party platforms from English-speaking countries.
- ▶ Each statement gets a CMP code, e.g. “decentralization”, “education”
  - ▶ 45 topics
    - ▶ some topics are somewhat esoteric, such as “marxist analysis”
  - ▶ We normalized to 19 broader, more interpretable topics

# Featurizing the Statements

- ▶ Each row includes a CMP code and a statement.
  - ▶ The statements are plain text.
- ▶ Standard featurization steps:
  - ▶ remove capitalization, punctuation, stopwords
  - ▶ construct n-grams up to length 3
  - ▶ remove n-grams appearing in less than 10 statements or more than 40 percent of statements
    - ▶  $M = 7,646$  features
  - ▶ compute tf-idf-weighted n-gram frequencies

# Regularized Logit Model

- ▶  $N$  rows,  $M$  text features,  $K$  policy topics
- ▶ Probability model:

$$P(Y_i = c) = \frac{e^{\beta_c X_i}}{\sum_{k=1}^K e^{\beta_k X_i}},$$

where  $c \in 1, \dots, K$  are the topic labels and  $\beta$  is an  $M \times K$  matrix of parameters.

- ▶ Cost function:

$$J(\beta) = -\frac{1}{N} \left[ \sum_{i=1}^N \sum_{k=1}^K \mathbf{1}\{y^i = k\} \log \frac{e^{\beta_k X_i}}{\sum_{l=1}^K e^{\beta_l X_i}} \right] + \gamma \sum_{j=1}^M \sum_{k=1}^K \beta_{jk}^2$$

- ▶  $\gamma$  = strength of L2 penalty
  - ▶  $\gamma^* = 1/2$  selected by 3-fold cross-validation grid search.

# Prediction Model

- ▶ Given a chunk of text, the logistic model computes a probability distribution over policy topics.
  - ▶ harnesses expert knowledge about political topics from Manifesto Project
- ▶ Validation of accuracy: predict the CMP code in a held-out sample of manifesto corpus statements
  - ▶ In-sample accuracy = 71%, Out-of-sample accuracy = 53%
    - ▶ quite good given there are 19 policy areas – choosing randomly would be correct 5% of the time; choosing top category (other topic) would be correct 15% of the time.



# Confusion Matrix

	Admin.	Agri.	Cult.	De-cent.	Econ.	Educ.	Freed.	Intl.	Labor	Milit.	Natl Way Life	Demos	Other	Party Pol.	Quality	Target	Tech	Morals	Welfare	Total True
Administration	270	5	2	7	77	12	71	7	5	3	4	1	78	11	36	3	11	0	101	704
Agriculture	5	114	0	3	24	3	7	4	2	0	2	2	23	1	56	0	7	0	22	275
Culture	8	3	155	5	22	21	22	4	0	1	19	4	56	3	18	7	17	1	26	392
Decentralization	27	2	6	73	16	11	24	1	0	1	3	1	32	7	20	0	13	2	22	261
Economics	59	12	5	4	715	13	29	16	19	2	11	4	138	30	96	1	49	2	126	1331
Education	7	1	9	1	17	461	8	0	0	0	1	4	75	10	10	2	37	3	69	715
Freedom	51	0	9	20	41	13	642	19	3	16	12	6	126	34	16	4	14	7	89	1122
Internationalism	8	1	4	3	33	2	45	245	2	14	11	1	46	9	19	3	3	0	26	475
Labor Groups	11	2	2	0	36	7	12	2	92	0	2	3	20	3	13	0	6	1	40	252
Military	7	0	1	0	7	1	24	27	2	118	5	0	32	2	10	0	9	1	13	259
Nat'l Way of Life	4	1	5	3	38	3	32	17	2	6	88	6	61	21	19	2	2	4	44	358
Non-Econ Demo Grps	7	1	7	0	15	15	30	1	5	2	5	95	36	8	4	5	5	5	104	350
Other Topic	38	14	17	13	78	48	86	25	20	12	20	26	1263	39	71	20	48	7	159	2004
Party Politics	20	5	5	2	44	2	48	9	4	5	10	0	80	183	25	4	10	2	53	511
Quality of Life	21	17	9	4	102	9	13	14	7	3	7	1	112	16	684	0	42	0	34	1095
Target Groups	11	2	7	2	7	17	24	2	4	0	8	1	28	4	1	67	8	1	63	257
Tech & Infra	22	8	5	8	57	31	6	6	5	1	4	2	85	1	62	1	413	0	40	757
Trad'l Morality	2	0	2	0	6	9	21	2	0	0	8	3	25	4	1	1	0	61	40	185
Welfare	33	6	11	3	93	32	62	3	10	1	9	26	142	37	29	11	23	7	1173	1711
Total Predicted	611	194	261	151	1428	710	1206	404	182	185	229	186	2458	423	1190	131	717	104	2244	