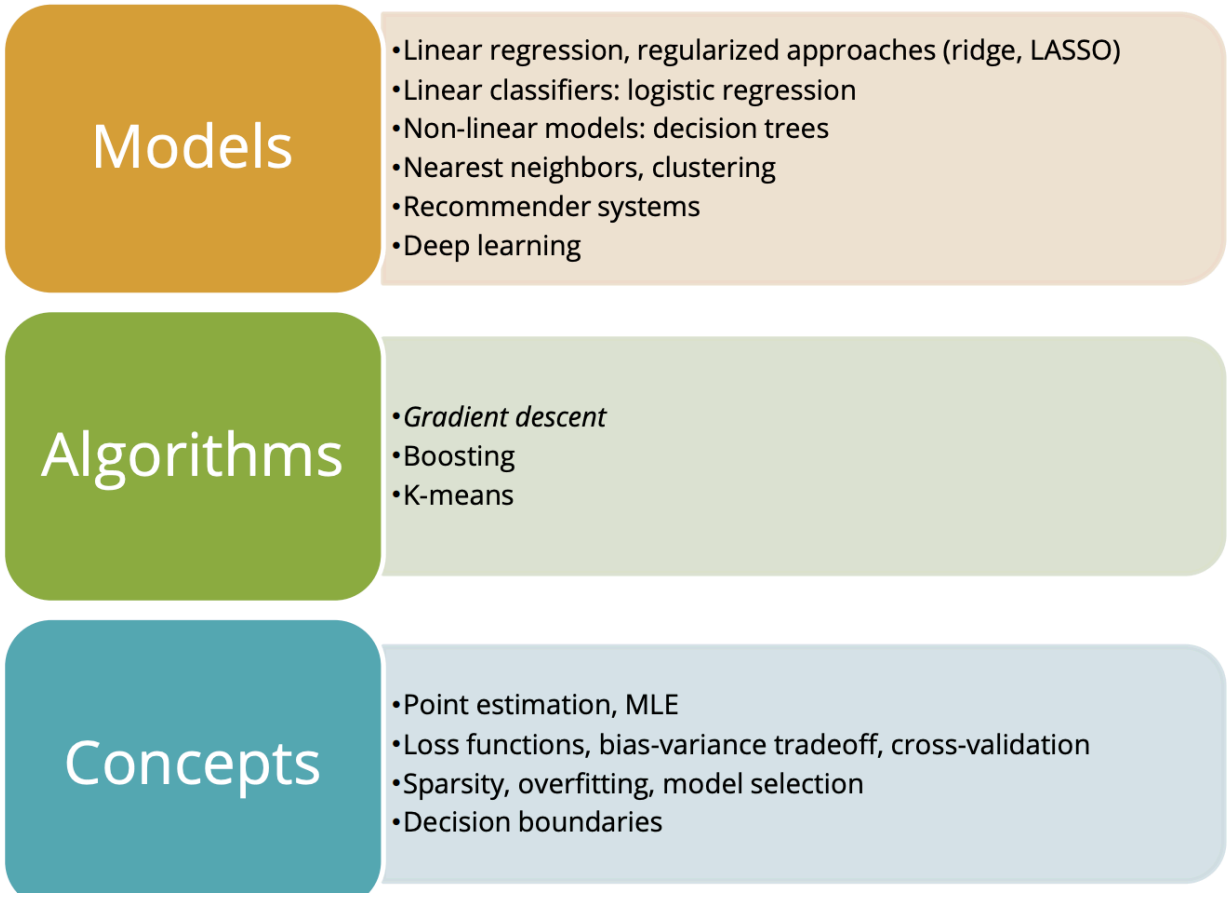
Final reflection (mind map in final page)

1. Summary:
   Things we learn this quarter:



2. Concepts:

   Models:
   Linear regression: use linear to fit data with Predictor:
   $$\widehat{w} = \min_{w} RSS(w)$$
   Regularized approach: ridge and Lasso and Elastic Net
   Ridge Predictor:
   $$\widehat{w} = \min_{w} RSS(W) + \lambda \|w\|_2^2$$
   Lasso Predictor (have 0-coef variable which can be discarded:
   $$\widehat{w} = \min_{w} RSS(W) + \lambda \left|\left|w\right|\right|_1$$
   Elastic Net:
   $$\widehat{w}_{ElasticNet} = \min_{w} RSS(w) + \lambda_1 \left|\left|w\right|\right|_1 + \lambda_2 \left|\left|w\right|\right|_2^2$$

Linear in code:

```
model = LinearRegression().fit(train_sales, train_price)
y_pred = model.predict(test_sales)
test_rmse = mean_squared_error(test_price, y_pred, squared=False)
```

Ridge in code:

```
r_model = Ridge(alpha=lamb[i], random_state=0)
r_model.fit(train_sales, train_price)
y_pred = r_model.predict(train_sales)
train_rmse = mean_squared_error(train_price, y_pred, squared=False)
```

Lasso in code:

```
l1_model = Lasso(alpha=lamb[i], random_state=0)
l1_model.fit(train_sales, train_price)
y_pred = l1_model.predict(train_sales)
train_rmse = mean_squared_error(train_price, y_pred, squared=False)
```

Logistic regression (sentiment analysis):

Give weights to words

```
# Note: C = 1/Lambda. Setting C to a really high value is the same as setting lambda = 0
sentiment_model = LogisticRegression(penalty='l2', random_state=1, C=1e23)
sentiment_model.fit(train_data[features], train_data['sentiment'])
```

Non-linear model: decision tree

For categorical variable, we need to use get dummies method.

```
loans = pd.get_dummies(loans)
clf = DecisionTreeClassifier(max_depth=6, random_state=6)
decision_tree_model = clf.fit(train_data[features], train_data[target])
```

Adaboost:

We train each model in succession, where we use the errors of the previous model to affect how we learn the next one.

```
adaboost_model = AdaBoostClassifier(n_estimators=100, random_state=0)
adaboost_model.fit(X_train[new_features],y_train)
y_pred = adaboost_model.predict(X_valid[new_features])
```

Nearest neighbors, clustering: knn

Use majority classifier to judge the property of a word by seeing the nearest k words.

```
knn15_model = KNeighborsClassifier(n_neighbors = 15, weights = 'distance')
knn15_model.fit(X_train[new_features], y_train)
```

Recommendation system:

```
vectorizer = TfidfVectorizer(max_df=0.95)  # ignore words with very high doc frequency
tf_idf = vectorizer.fit_transform(text['text'])
```

```
# TODO create and fit the model and transform our data
nmf = NMF(n_components=5, init='nndsvd', random_state=1)
tweets_projected = nmf.fit_transform(tf_idf)
```

```
# TODO find index of largest topic
largest_topic = np.bincount(tweets_projected.argmax(axis = 1)).argmax()
```

Deep learning (use of cnn net):

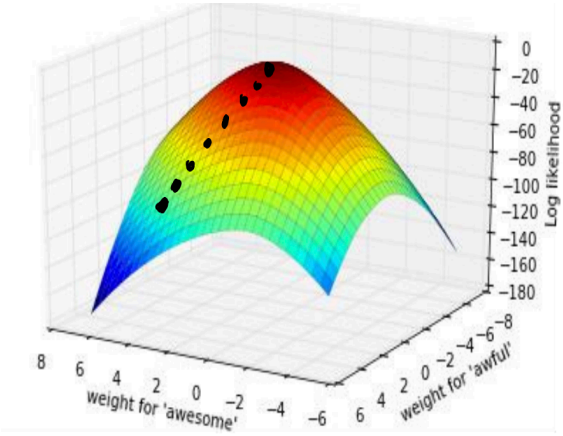
```
class NetD(nn.Module):
    def __init__(self):
        super(NetD, self).__init__()
        self.conv1 = nn.Conv2d(3, 50, 3)
        self.conv2 = nn.Conv2d(50, 300, 2)

        self.fc1 = nn.Linear(300*7*7, 500)
        self.fc2 = nn.Linear(500, 80)
        self.fc3 = nn.Linear(80, NUM_CLASSES)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2) #50@15*15
        x = F.max_pool2d(F.relu(self.conv2(x)), 2) #300@7*7
        x = torch.flatten(x, 1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Algorithm:

Gradient accent:

gradient ascent!

$$\hat{w} = \underset{w}{\operatorname{argmax}} \prod_{i=1}^{n} P(y_i|x_i, w)$$



Gradient decent is in opposite way.

Random forest (bagging):

Training

- Make T random samples of the training data that are the same size as the training data but are sampled with replacement
- Train a really tall tree on each sampled dataset (overfit)

Predict

- For a given example, ask each tree to predict what it thinks the label should be
- Take a majority vote over all trees

Adaboost (boosting):

We train each model in succession, where we use the errors of the previous model to affect how we learn the next one.

To do this, we will need to keep track of two types of weights

- The first are the $\hat{w}_t$ that we will use as the end result to weight each model.
  - Intuition: An accurate model should have a high weight
- We will also introduce a weight α_i for each example in the dataset that we update each time we train a new model
  - Intuition: We want to put more weight on examples that seem hard to classify correctly

k-means: clustering (unsupervised)

Step 0: Initialize cluster centers

Repeat until convergence:

Step 1: Assign each example to its closest cluster centroid

Step 2: Update the centroids to be the average of all the points assigned to that cluster

Point estimation, MLE (maximize likelihood estimate)

Find the w that maximizes the likelihood

$$\hat{w} = \underset{w}{\operatorname{argmax}}\, \ell(w) = \underset{w}{\operatorname{argmax}} \prod_{i=1}^{n} P(y_i|x_i, w)$$

Generally we maximize the log-likelihood which looks like

$$\hat{w} = \underset{w}{\operatorname{argmax}}\, \ell(w) = \underset{w}{\operatorname{argmax}} \log(\ell(w)) = \underset{w}{\operatorname{argmax}} \sum_{i=1}^{n} \log(P(y_i|x_i, w))$$

Also commonly written by separating out positive/negative terms

$$\hat{w} = \underset{w}{\operatorname{argmax}} \sum_{i=1:y_i=+1}^{n} \ln\left(\frac{1}{1+e^{-w^T h(x)}}\right) + \sum_{i=1:y_i=-1}^{n} \ln\left(\frac{e^{-w^T h(x)}}{1+e^{-w^T h(x)}}\right)$$

Bias-variance trade off:

$$Error = Bias^2 + Variance + Noise$$

High complexity has high variance

Notation

- $C_{TP} = \#\text{TP}, \quad C_{\text{FP}} = \#\text{FP}, \quad C_{\text{TN}} = \#\text{TN}, \quad C_{\text{FN}} = \#\text{FN}$
- $N = C_{TP} + C_{FP} + C_{TN} + C_{FN}$
- $N_P = C_{TP} + C_{FP}, \quad N_N = C_{FP} + C_{TN}$

Error Rate

$$\frac{C_{FP} + C_{FN}}{N}$$

Accuracy Rate

$$\frac{C_{TP} + C_{TN}}{N}$$

False Positive rate (FPR)

$$\frac{C_{FP}}{N_N}$$

False Negative Rate (FNR)

$$\frac{C_{FN}}{N_P}$$

True Positive Rate or Recall

$$\frac{T_P}{N_P}$$

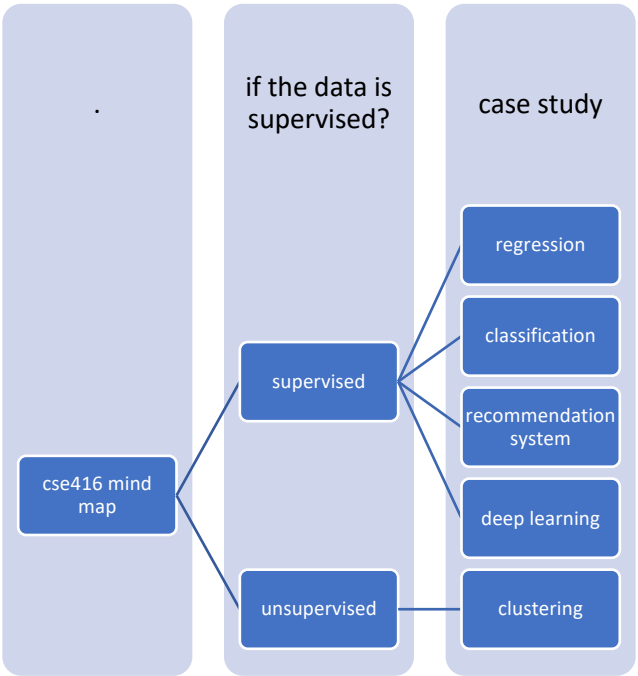
Precision

$$\frac{T_P}{C_{TP} + C_{FP}}$$

F1-Score

$$2\frac{Precision \cdot Recall}{Precison + Recall}$$

See more!



I don’t know how to add an extra column to the left, so I wrote below:

Regression: use linear to approximate data
Classification: judge if a sentence is good or bad

Recommendation system: recommend by known user habits
Deep learning: identify the object by seeing the feature of what most of such object looks like.
Clustering: cluster similar data.

3. Uncertainty:
   I think there is no so confuse concept during this class maybe. Thanks for the whole quarter. I am very appreciate the required learning reflection.