Lecture 3
K-Nearest Neighbors & Naïve-Bayes Classifiers

Wonjun Lee

Contents

- K-Nearest Neighbors
- Naïve-Bayes Classifiers

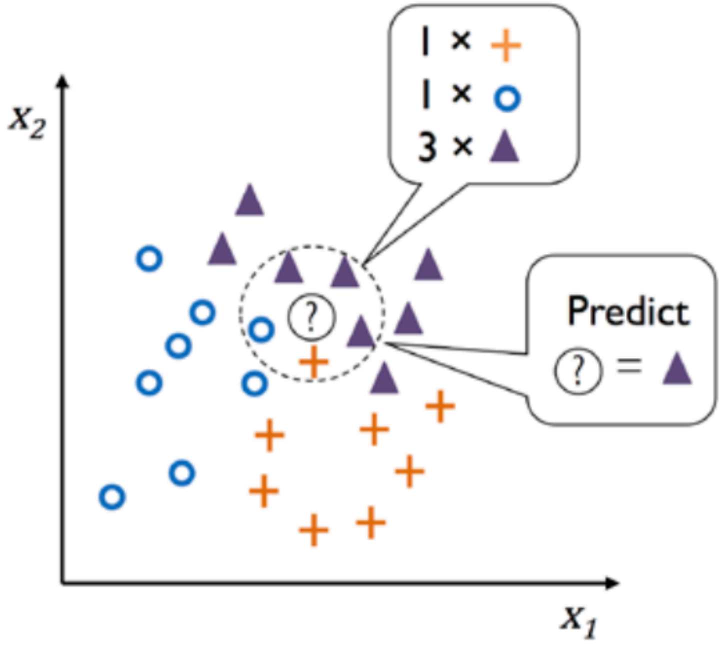
K-Nearest Neighbors

K-Nearest Neighbors (KNN)

- Intuition – Similar things are near to each other
- Lazy learner – does not learn a discriminative function from training data but memorizes training dataset instead
- Instance-based learning – subcategory of nonparametric models
  - ✓ Parametric – Estimate parameters from the training dataset to learn a function that can classify new data points without requiring the original training dataset anymore. Ex) Perceptron, logistic regression, linear SVM
  - ✓ Nonparametric – the complexity of the model grows with the number of training data. Ex) Decision tree, random forest, kernel SVM

KNN Algorithm

1. Choose *k* and distance metric
2. Find *k*-nearest neighbors of the data record that we want to classify
   ✓Based on the chosen distance metric, the KNN algorithm finds the *k* examples in the training dataset that are closest (most similar) to the point that we want to classify
3. Assign the class label by majority vote
   ✓The class label of the data point is then determined by a majority vote among its *k* nearest neighbors



KNN Algorithm

http://127.0.0.1:8888/notebooks/work/python-machine-learning-book-3rd-edition-master/ch03/knn.ipynb

```
def knn(data, query, k, distance_fn, choice_fn):
    neighbor_distances_and_indices = []

    # 3. For each example in the data
    for index, example in enumerate(data):
        # 3.1 Calculate the distance between the query example and the current
        # example from the data.
        distance = distance_fn(example[:-1], query)

        # 3.2 Add the distance and the index of the example to an ordered collection
        neighbor_distances_and_indices.append((distance, index))

    # 4. Sort the ordered collection of distances and indices from
    # smallest to largest (in ascending order) by the distances
    sorted_neighbor_distances_and_indices = sorted(neighbor_distances_and_indices)

    # 5. Pick the first K entries from the sorted collection
    k_nearest_distances_and_indices = sorted_neighbor_distances_and_indices[:k]

    # 6. Get the labels of the selected K entries
    k_nearest_labels = [data[i][-1] for distance, i in k_nearest_distances_and_indices]

    # 7. If regression (choice_fn = mean), return the average of the K labels
    # 8. If classification (choice_fn = mode), return the mode of the K labels
    return k_nearest_distances_and_indices , choice_fn(k_nearest_labels)
```

Advantage/Disadvantage of a Memory-based Approach

- (+) Classifier immediately adapts as we collect new training data
- (-) Computational complexity grows linearly with # of examples in training dataset
- (-) Storage space becomes a challenge when working with large datasets

Code

- http://127.0.0.1:8888/notebooks/work/python-machine-learning-book-3rd-edition-master/ch03/ch03.ipynb#K-nearest-neighbors---a-lazy-learning-algorithm

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=5,
                           p=2,
                           metric='minkowski')
knn.fit(X_train_std, y_train)

plot_decision_regions(X_combined_std, y_combined,
                      classifier=knn, test_idx=range(105, 150))

plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.tight_layout()
#plt.savefig('images/03_24.png', dpi=300)
plt.show()
```

Why do we need to fit a KNN classifier?

- Evaluating a KNN classifier on a new data point requires searching for its nearest neighbors in the training set, which can be an expensive operation when the training set is large
- Various tricks to speed up this search
  - ✓ Create various data structures based on the training set
  - ✓ Computational work needed to classify new points is actually common → this work can be done ahead of time and then re-used, rather than repeated for each new instance
  - ✓ Ex) construct *kd-trees* or *ball trees* during fit function

Resolving Ties

- In the case of a tie, the scikit-learn implementation of the KNN algorithm will prefer the neighbors with a closer distance to the data record to be classified
- If the neighbors have similar distances, the algorithm will choose the class label that comes first in the training dataset

Choosing the Right *k*

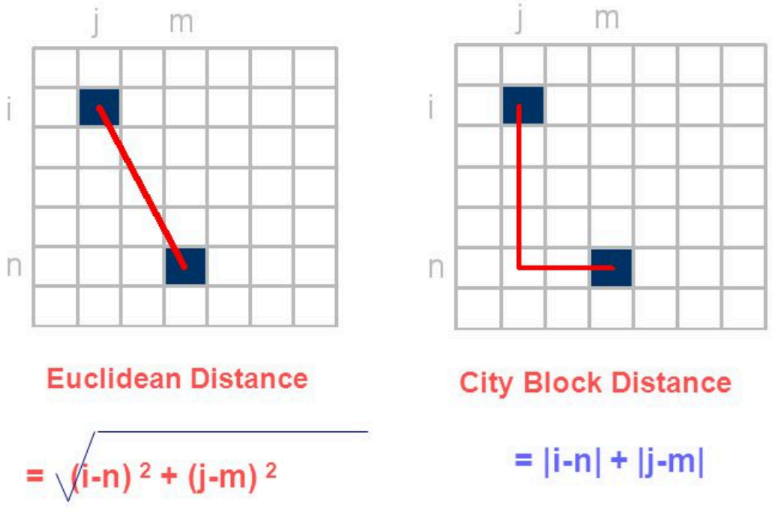
- The right choice of k is crucial for a good balance between overfitting and underfitting
- *k* = 1 : Predictions become less stable
  - ✓ A query point surrounded by several reds and one green, but the green is the single nearest neighbor → KNN incorrectly predicts that the query point is green
- *k* → high value : Predictions become more stable due to majority voting → more accurate up to a certain point
- Usually make *k* an odd number to have a tiebreaker

Distance Metric

- Euclidean distance is simple
  - ✓Importance to standardize the data so that each feature contributes equally to the distance
  - ✓*minkowski* – generalization of the Euclidean and Manhattan distance

$$d\left(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\right) = \sqrt[p]{\sum_k \left|x_k^{(i)} \quad x_k^{(j)}\right|^p},$$

p = 2 → Euclidean, p = 1 → Manhattan (city block)



The Curse of Dimensionality

- KNN is very susceptible to overfitting due to the curse of dimensionality
- The curse of dimensionality – phenomenon where the feature space becomes increasingly sparse for an increasing number of dimensions of a fixed-size training dataset
- To avoid – use feature selection and dimensionality reduction

Code Examples

- http://127.0.0.1:8888/notebooks/work/python-machine-learning-book-3rd-edition-master/ch03/knn.ipynb

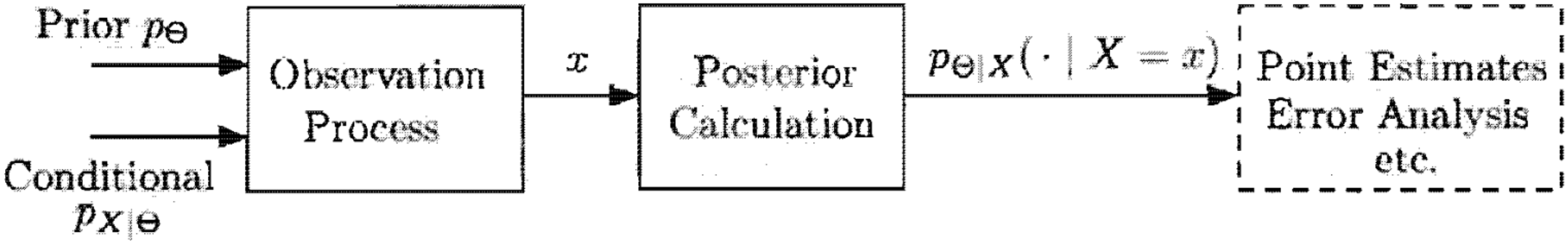
Naïve-Bayes Classifiers

Overview

- Linear classifiers
- Based on Bayes’ theorem
- Naïve – assumption that features are mutually independent
- Tend to perform very well under this assumption – especially for small sample sizes
- (+) diagnosis of diseases, Classification of RNA sequences, spam filtering
- (-) very poor performance on problems with strong violations of independence assumption and non-linear classification problems

Bayes' Rule

- Once a particular value x of X has been observed, a complete answer to the Bayesian inference problem is provided by the posterior distribution $p_{\Theta|X}(\theta|x)$ or $f_{\Theta|X}(\theta|x)$ of Θ

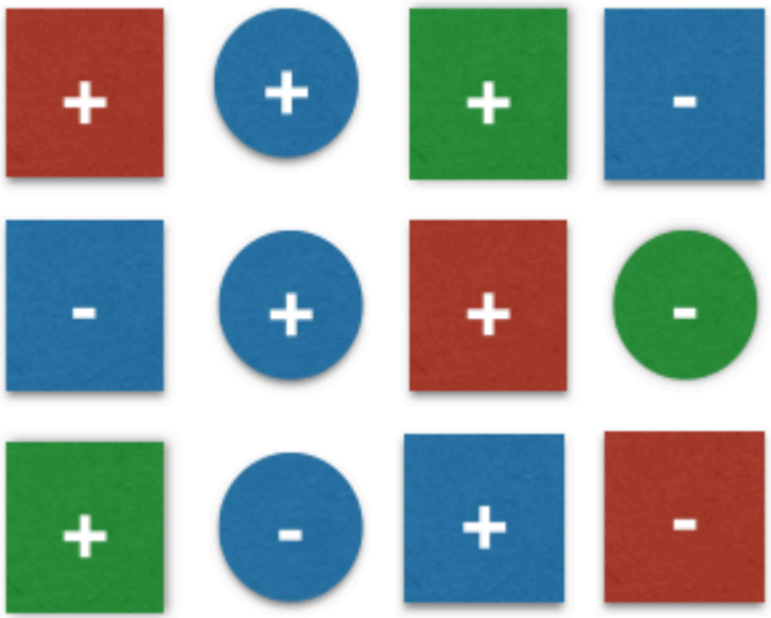


- Θ discrete, X discrete:

$$p_{\Theta|X}(\theta \mid x) = \frac{p_\Theta(\theta)p_{X|\Theta}(x \mid \theta)}{\sum_{\theta'} p_\Theta(\theta')p_{X|\Theta}(x \mid \theta')} \equiv posterior\ probability = \frac{prior\ probability \cdot conditional\ probability}{evidence}$$

Naïve Bayes – A Toy Example

- 12 samples
- 2 features = color and shape
- 2 classes = + and –
- θ : class labels
  - ✓ $\theta_i = \{+, -\}$
- $x_i = [x_{i1}, x_{i2}]$: 2 dim feature vectors
  - ✓ $x_{i1} \in$ {blue, green, red, yellow}
  - ✓ $x_{i2} \in$ {circle, square}
  - ✓ for $i \in \{1, 2, \ldots, 12\}$
- Task: classify a new sample



Naïve Bayes – Maximum-Likelihood Estimates

- Decision Rule : Classify sample as + if

$$P(\theta = + \mid x = [\text{blue, square}]) \geq P(\theta = - \mid x = [\text{blue, square}])$$

Else classify sample as −

- Under the assumption that the samples are i.i.d, the prior probabilities can be obtained via the maximum-likelihood estimate:
- $P(+|x) : P(x|+)P(+) = \{P(\text{blue}|+)P(\text{square}|+)\}\ P(+) = \{(\frac{3}{7})(\frac{5}{7})\}(\frac{7}{12}) = (\frac{5}{28}) = 0.18$
- $P(-|x) : P(x|-)P(-) = \{P(\text{blue}|-)P(\text{square}|-)\}\ P(-) = \{(\frac{3}{5})(\frac{3}{5})\}(\frac{5}{12})=(\frac{3}{20})=0.15$

Naïve Bayes – A Toy Example

- Classification
  - ✓Decision rule: If P(+|x) > P(−|x)
    classify as "+",
    else classify as "−"
  - ✓0.18 > 0.15 → sample is classified as "+"

Naïve Bayes – Additive Smoothing

- What about a new value for the color attribute that is not present in the training dataset, e.g.,

(class "**+**" and features x = [yellow, square])?

- $P(x|+) = P(yellow|+)P(square|+) = 0*(\frac{5}{7}) = 0 \rightarrow P(+|x) = 0$
- $P(x|-)= P(yellow|-)P(square|-) = 0*(\frac{3}{5}) = 0 \rightarrow P(-|x) = 0$

Naïve Bayes – Additive Smoothing

- To avoid zero probabilities : add smoothing term to multinomial Bayes model
  - ✓Lidstone smoothing : α < 1
  - ✓Laplace smoothing : α = 1

$$\hat{P}(x_i|\theta_j) = \frac{N_{x_i,\theta_j}+\alpha}{N_{\theta_j}+\alpha d} \text{ (i = 1, 2, ..., d)}$$

  - ✓N_{x_i,θ_j} : # of times feature x_i appears in samples from class θ_j
  - ✓N_{θ_j} : Total count of all features in class θ_j
  - ✓α : Parameter for additive smoothing
  - ✓d : Dimensionality of the feature vector x = [x_1, …, x_d]

Naïve Bayes – Additive Smoothing

- Suppose α = 0.5
- P(+|x) : P(x|+)P(+) = {P(yellow|+)P(square|+)} P(+) = $(\frac{0+0.5}{7+0.5*2})(\frac{5+0.5}{7+0.5*2})(\frac{7}{12})$ = 0.025
- P(−|x) : P(x|−)P(−) = {P(yellow|−)P(square|−)} P(−) = $(\frac{0+0.5}{5+0.5*2})(\frac{3+0.5}{5+0.5*2})(\frac{5}{12})$ = 0.02

From $\hat{P}\left(x_i \middle| \theta_j\right) = \frac{N_{x_i,\theta_j}+\alpha}{N_{\theta_j}+\alpha d}$ (i = 1, 2, …, d)

➔ P(+|x) > P(−|x)

Naïve Bayes with a Single Feature – Weather Example

- Play when outlook = 'overcast'?
- P(Yes|overcast) = P(Yes)P(overcast|Yes) = $(\frac{9}{14})(\frac{4}{9})$ = 0.29
- P(No|overcast) = P(No)P(overcast|No) = $(\frac{5}{14})(\frac{0}{5})$ = 0

→ P(Yes|overcast) > P(No|overcast)

outlook	Yes	No	
overcast	4		4/14
sunny	2	3	5/14
rainfall	3	2	5/14
	9/14	5/14	

Day	outlook	Decision
1	sunny	No
2	sunny	No
3	overcast	Yes
4	rainfall	Yes
5	rainfall	Yes
6	rainfall	No
7	overcast	Yes
8	sunny	No
9	sunny	Yes
10	rainfall	Yes
11	sunny	Yes
12	overcast	Yes
13	overcast	Yes
14	rainfall	No

Naïve Bayes with Multiple Features – Weather Example

- Play when outlook = ‘overcast’ and temperature = ‘mild’?
- P(Yes|{outlook=overcast},{temp=mild}) = P(Yes)P({outlook=overcast},{temp=mild}|Yes) = $(\frac{9}{14})(\frac{16}{81})$ = 0.13
  - ✓ P({outlook=overcast},{temp=mild}|Yes)= P({outlook=overcast}|Yes) P({temp=mild}|Yes) = $(\frac{4}{9})$ $(\frac{4}{9})$ = $(\frac{16}{81})$
- P(No|{outlook=overcast},{temp=mild}) = P(No)P({outlook=overcast},{temp=mild}|No) = $(\frac{5}{14})$(0) = 0
  - ✓ P({outlook=overcast},{temp=mild}|No)= P({outlook=overcast}|No) P({temp=mild}|No) = (0) $(\frac{2}{5})$ = (0)

→ P(Yes|{outlook=overcast},{temp=mild}) > P(No|{outlook=overcast},{temp=mild})

Day	outlook	temperature	Decision
1	sunny	hot	No
2	sunny	hot	No
3	overcast	hot	Yes
4	rainfall	mild	Yes
5	rainfall	cool	Yes
6	rainfall	cool	No
7	overcast	cool	Yes
8	sunny	mild	No
9	sunny	cool	Yes
10	rainfall	mild	Yes
11	sunny	mild	Yes
12	overcast	mild	Yes
13	overcast	hot	Yes
14	rainfall	mild	No

Naïve Bayes with Multiple Features – Weather Example

- Code:

http://127.0.0.1:8888/notebooks/work/python-machine-learning-book-3rd-edition-master/ch03/naive_bayes_classifier.ipynb

Naïve Bayes with Multiple Labels – Wine Example

- Multi-class classification problem
- Dataset – “result of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars” (UC Irvine Machine Learning Repository)
- *from sklearn.naive_bayes import GaussianNB* – when the features have continuous values

Naïve Bayes with Multiple Labels – Wine Example

- Code:

http://127.0.0.1:8888/notebooks/work/python-machine-learning-book-3rd-edition-master/ch03/naive_bayes_classifier.ipynb