Multiclass, Bias and Variance

Ludwig Krippahl
Summary

- Multiclass classification
- Bias: deviation from expected value
- Variance: scattering of predictions
- Computing Bias and Variance with bootstrapping
- Relation to underfitting and overfitting
Multiclass Classification
Multiclass Classification

Some classifiers naturally handle multiple classes

- k-NN: output majority class from k neighbours.
- Naïve Bayes: output class with greatest joint probability

\[ C^{NaïveBayes} = \arg\max_{k \in \{0,1,\ldots,K\}} \ln p(C_k) + \sum_{j=1}^{N} \ln p(x_j|C_k) \]
Example:


- 3 Classes
  - Setosa
  - Versicolor
  - Virginica

- 4 Attributes:
  - Sepal length
  - Sepal width
  - Petal length
  - Petal width
Iris dataset

CC BY-SA Setosa: Szczecinkowaty; Versicolor: Gordon, Robertson; Virginica: Mayfield
Multiclass Classification

- Example: Iris dataset
Multiclass Classification

k-NN classification (k = 15)
Multiclass Classification

Linear discriminant classifiers need some adaptation

\[ \vec{w}^T \vec{x} + w_0 = 0 \]

- (Logistic Regression, SVM, MLP...)

**Generic solutions for binary classifiers:**

- One versus the rest: K-1
- One versus one: K(K-1)/2
- One versus the rest: K, max
Multiclass Classification

Multiclass: one vs the rest (K-1)

- Create K-1 classifiers
- For each k in \{1 \ldots K-1\}, set class k as 1 and others as 0
- Assign class \{1 \ldots K-1\} according to which classifier returns 1, or class K if none.
Multiclass Classification

- One vs the rest (K-1), Classifier for Setosa
One vs the rest (K-1), Classifier for Versicolor
Multiclass Classification

- One vs the rest (K-1), Final classifier (ambiguous areas)
Multiclass: one vs one $K(K-1)/2$

- Build classifiers for all pairs.
Multiclass Classification

- One vs one $K(K-1)/2$, Setosa vs Versicolor
Multiclass Classification

- One vs one $K(K-1)/2$, Setosa vs Virginica
Multiclass Classification

- One vs one $K(K-1)/2$, Versicolor vs Virginica
Multiclass Classification

- One vs one $K(K-1)/2$, Final: also ambiguous areas
Multiclass Classification

Multiclass: one vs rest K (or one vs all)

- Better: K OvR classifiers
Multiclass Classification

- One vs rest: Classify using max of decision function
Pros of multiclass classification with one vs rest:

- Classify using max of decision function
- Helps avoid ambiguous classifications (depending on decision function)

Cons of multiclass classification with one vs rest:

- Classifiers may be unbalanced (more negative than positive)
- The confidence values of the decision function may not be directly comparable
Multiclass Classification

Logistic Regression

- Extend cross-entropy error function:

\[
p(T|w_1, \ldots, w_K) = \prod_{n=1}^{N} \prod_{k=1}^{K} p(C_k|\phi_n)^{t_{nk}} = \prod_{n=1}^{N} \prod_{k=1}^{K} y_{nk}^{t_{nk}}
\]

\[
E(w_1, \ldots, w_K) = - \sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \ln y_{nk}
\]

- Where \( t_{nk} \) is 1 if the point is in class \( k \), 0 otherwise

```python
from sklearn.linear_model import LogisticRegression

# One versus rest, max
logreg = LogisticRegression(C=1e5, multi_class='ovr')
logreg.fit(X, Y)

# Cross entropy
logreg = LogisticRegression(C=1e5, multi_class='multinomial')
logreg.fit(X, Y)
```
Multiclass Classification

Multilayer Perceptron

- For 2 classes, need only one output neuron
  - Sigmoid, probability of belonging to $C_1$
- For K classes, use K output neurons with softmax function:
  - (extension of sigmoid)

\[
\sigma : \mathbb{R}^K \rightarrow [0, 1]^K \\
\sigma(\bar{x})_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}
\]

- Softmax returns a vector where $\sigma_j \in [0, 1]$ and $\sum_{k=1}^{K} \sigma_k = 1$
- Represents probability of example belonging to each class $C_j$
General solution: One vs Rest

OneVsRestClassifier

- Fits one classifier per class, calling fit()
- Predicts from maximum of decision_function()

```python
ovr = OneVsRestClassifier(SVC(kernel='rbf', gamma=0.7, C=10))
ovr.fit(X, Y)
ovr.predict(test_set)
```
Multiclass Classification

- OneVsRestClassifier: automate OvR, max. decision
Suppose the model cannot fit the data
When we average over different samples...
...there is a large [bias] in the prediction
Bias: deviation of the average estimate from the target value
Bias: deviation of the average estimate from the target value

The bias for example $n$ is the squared error between the true value for $n$ and the average of the predictions for $n$, over all hypotheses trained on different samples:

$$bias_n = (\bar{y}(x_n) - t_n)^2$$

The bias for the model is the average bias for all examples:

$$bias = \frac{1}{N} \sum_{n=1}^{N} (\bar{y}(x_n) - t_n)^2$$

Note: the bias is often written as $bias^2$, but this is just to denote the squared error.
Variance
If the model is overfitting, it adjusts the training data too much.
Hypothesis varies over different training sets
Variance: dispersion of predictions around their average
Variance: dispersion of predictions around their average

Variance of predictions for point \( n \), over all hypotheses:

\[
\text{var}_n = \frac{1}{M} \sum_{m=1}^{M} (\bar{y}(x_n) - y_m(x_n))^2
\]

(Average square dist. to average)

Variance for the model is the average over all points

\[
\text{var} = \frac{1}{NM} \sum_{n=1}^{N} \sum_{m=1}^{M} (\bar{y}(x_n) - y_m(x_n))^2
\]
Bias and Variance

Bias: squared deviation from true value

\[
bias = \frac{1}{N} \sum_{n=1}^{N} (\bar{y}(x_n) - t_n)^2
\]

Variance: squared deviation from mean prediction

\[
var = \frac{1}{NM} \sum_{n=1}^{N} \sum_{m=1}^{M} (\bar{y}(x_n) - y_m(x_n))^2
\]

Note: Bias and Variance depend on what the model does on average and not on any single hypothesis.
Bias-variance decomposition
Bias-variance decomposition

- If we have a squared loss function, then the expected error is:
  \[
  E((y - t)^2) = (E(y) - E(t))^2 + E((y - E(y))^2) + E((t - E(t))^2) \\
  \text{bias} + \text{var} + \text{noise}
  \]

Bias-variance decomposition

- How to compute: average over different training sets, evaluate outside training set.
- But we have only one training set
- We need a resampling method
Bias-variance decomposition

Bootstrapping
- From the training set, sample at random with replacement
- Generate M replicas of N points each
- Measure over the replicas

We also want an error estimate
- To get unbiased estimates of the errors, we need to measure it outside the training set.
- We can use a test (or validation) set
Bootstrapping

```python
def bootstrap(samples, data):
    train_sets = np.zeros((samples, data.shape[0], data.shape[1]))
    for sample in range(samples):
        ix = np.random.randint(data.shape[0], size=data.shape[0])
        train_sets[sample, :] = data[ix, :]
    return train_sets
```

- Training sets all have the same size, N, sampled with reposition
Bias-variance decomposition

Polynomial regression

```python
def bv_poly(degree, train_sets, test_set):
    samples = train_sets.shape[0]
predicts = np.zeros((samples, test_set.shape[0]))
    for ix in range(samples):
        coefs = np.polyfit(train_sets[ix,:,0],
                           train_sets[ix,:,1],degree)
        predicts[ix,:] = np.polyval(coefs, test_set[:,0])

    mean_preds = np.mean(predicts, axis=0)
    bias_per_point = (mean_preds - test_set[:, -1])**2
    bias = np.mean(bias_per_point)

    var_per_point = np.mean((predicts - mean preds)**2, axis=0)
    var = np.mean(var_per_point)

    return bias, var
```
Bias-variance decomposition, polynomial regression
Bias-variance decomposition

- Lowest total error
B-V with classifiers

- With a 0/1 loss function, the main prediction for point \( i \) is the mode.
- Assuming there is no noise, the Bias for point \( i \) is:
  \[
  bias_i = L(Mo(y_{i,m}), t_i) \quad bias_i \in \{0, 1\}
  \]
- And the Variance is:
  \[
  var_i = E \left( L(Mo(y_{i,m}), y_{i,m}) \right)
  \]
- To compute total error, we must consider that:
  - If \( bias_i = 0 \), \( var_i \) increases error.
  - If \( bias_i = 1 \), \( var_i \) decreases error.
- So for the error we must add or subtract the variances accordingly:
  \[
  E \left( L(t, y) \right) = E \left( B(i) \right) + E \left( V_{unb.}(i) \right) - E \left( V_{biased}(i) \right)
  \]
Bias-variance decomposition

KNN, Optimize neighbours
def bv_knn(neighs, train_sets, test_set):
    samples = train_sets.shape[0]
predicts = np.zeros((samples, test_set.shape[0]))
for ix in range(samples):
    sv = KNeighborsClassifier(n_neighbors=neighs)
    sv.fit(train_sets[ix,:,:-1], train_sets[ix,:,-1])
predicts[ix,:] = sv.predict(test_set[:,:-1])
main_preds = np.round(np.mean(predicts, axis=0))
bias_per_point = np.abs(test_set[:,-1]-main_preds)
var_per_point = np.mean(np.abs(predicts-main_preds), axis=0)
u_var = np.sum(var_per_point[bias_per_point == 0])/test_set.shape[0]
b_var = np.sum(var_per_point[bias_per_point == 1])/test_set.shape[0]
print(u_var, b_var)
return bias, u_var-b_var
Bias-variance decomposition

- Bias-variance decomposition with KNN
Bias-variance KNN, lowest estimated error (apart from noise)
Bias-variance tradeoff

- In general, reducing *bias* increases *variance* and vice-versa

Note: Bias-Variance decomposition is useful for understanding the components of the error but, in practice, it is easier to use cross-validation and just consider the total error.
Summary
Summary

- Bias: average deviation from true value
- Variance: dispersion around the average prediction
  - Classification: variance increases or decreases error depending on bias
- Bias and variance related to underfitting and overfitting

Further reading

- Alpaydin, Section 4.3
- Bishop, Sections 4.1.2, 4.3.4, 7.1.3
- Optional: