



Diabetes Prediction using Machine Learning Algorithms

- Sunny Patel (smp222@njit.edu)



Dataset

- UCI machine learning repository
- 7 features : Glucose, Blood Pressure, Skin Thickness, Insulin, BMI, Diabetes Pedigree Function, Age
- Boolean outcome



Implementation

- K – nearest neighbors
- Logistic regression
- Decision Tree
- Random forest
- Support vector machine



K-Nearest Neighbors

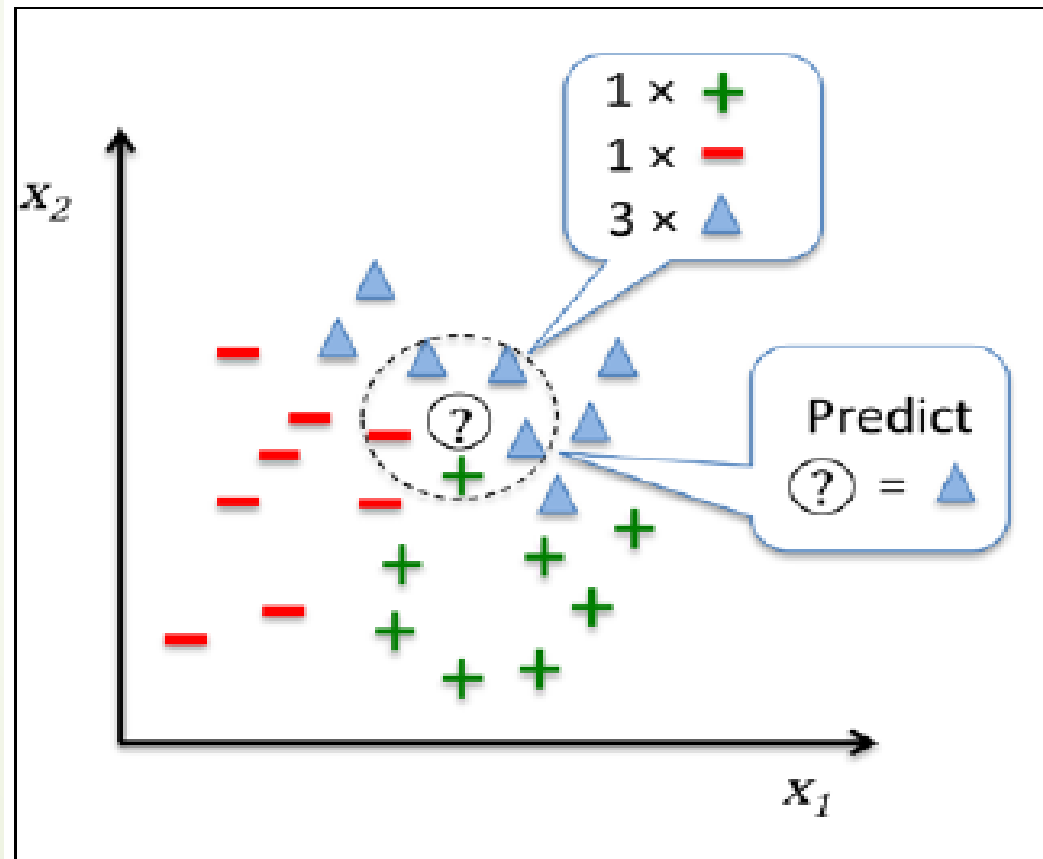
- The KNN algorithm finds the k samples in the training dataset that are closest (most similar) to the point that we want to classify. The class label of the new data point is then determined by a majority vote among its k nearest neighbors.
- Euclidean Distance metric to calculate distance between two points
- Choice of K is crucial if it is too low then it results in underfitting if it is too high then it results into overfitting.



K-Nearest Neighbors(cont.)

- Two approaches to calculate distances:
 - 1. brute force: calculate distance to all training data and then do sorting (better for sparse and small data)
 - 2. Consider a circle of radius r around data point and increase the r until we have desired no. of votes (better for dense data)
- Steps:
 - Choose the number of k and a distance metric.
 - Find the k nearest neighbors of the sample that we want to classify.
 - Assign the class label by majority vote.

K-Nearest Neighbors(cont.)



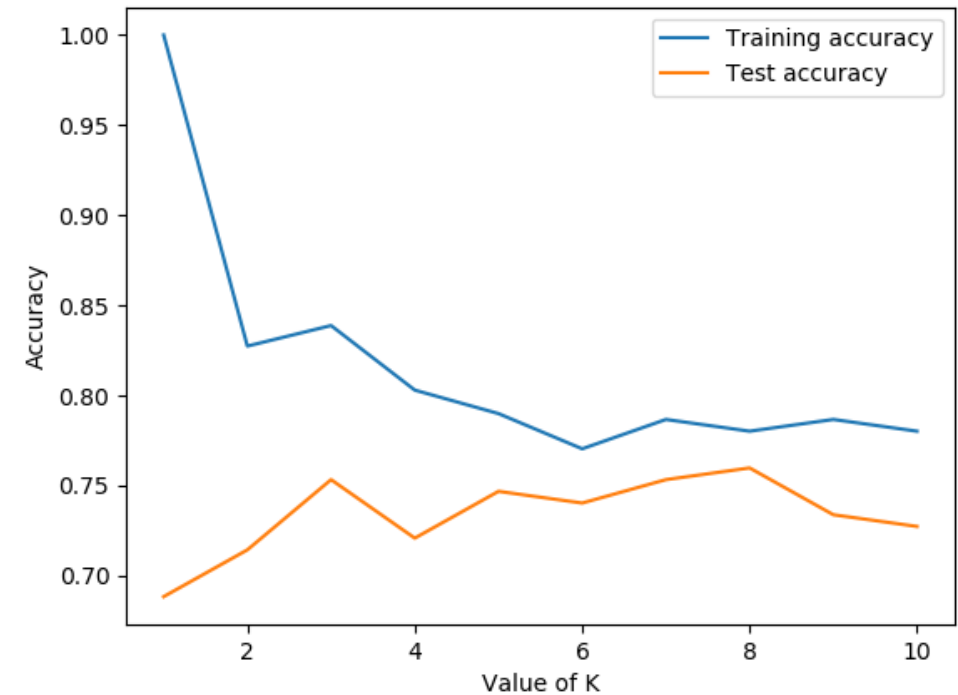


K-Nearest Neighbors(cont.)

- Drawbacks
 - Requires more memory as it “remembers” all training examples.
 - Doesn't work well for large datasets.

K-Nearest Neighbors(cont.)

| No of neighbors | Training accuracy | Test accuracy |
|-----------------|-------------------|-----------------|
| 1 | 1.000000 | 0. 649351 |
| 2 | 0.85668 | 0.65584 |
| 3 | 0.8599 | 0.636364 |
| 4 | 0.809446 | 0.675325 |
| 5 | 0.793160 | 0.675325 |
| 6 | 0.796417 | 0.707792 |
| 7 | 0.791531 | 0.720779 |
| 8 | 0.793160 | 0.707792 |
| 9 | 0.799674 | 0.707792 |
| 10 | 0.780130 | 0.727273 |

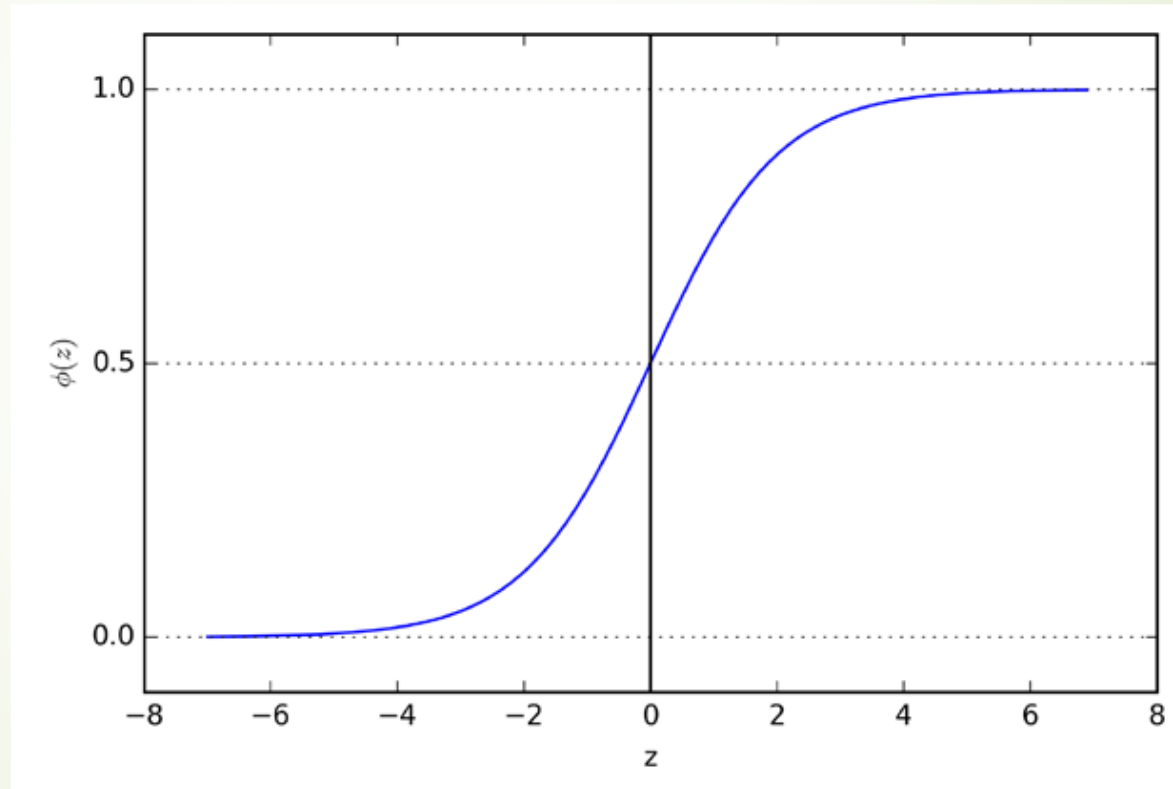




Logistic Regression

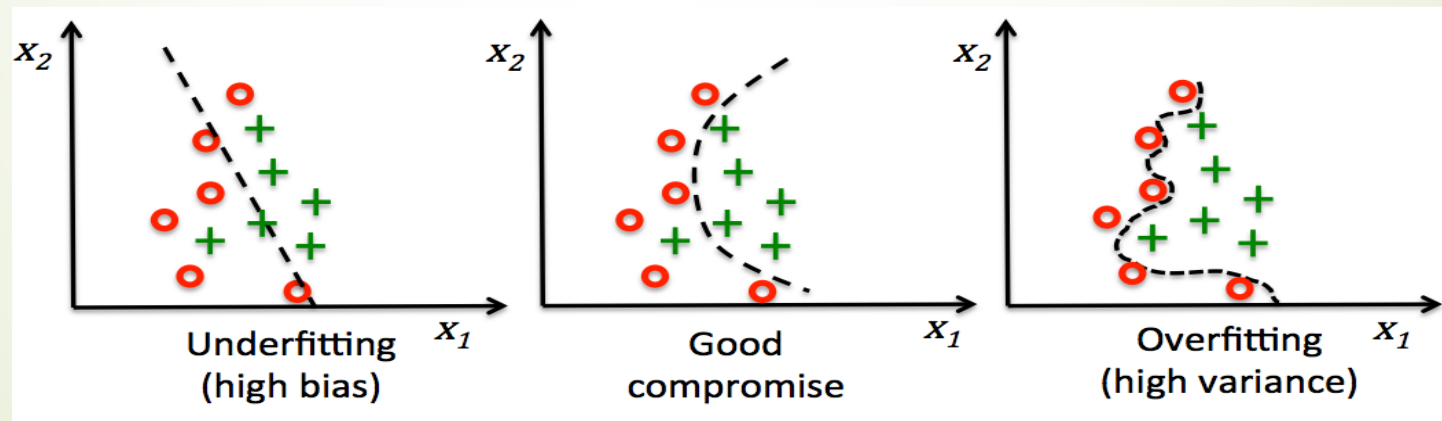
- Performs well on linearly separable data.
- $\text{Logit}(P) = \log(p / (1-p))$ where p = probability of true outcome
- The logit function takes input values in the range 0 to 1 and transforms them to values over the entire real number range, which we can use to express a linear relationship between feature values and the log-odds
- Logistic (Sigmoid) function (Inverse form of logit) $f(z): 1 / (1 + e^{-z})$ where z is linear combination of weights and sample features

Logistic Regression (cont.)



Logistic Regression (cont.)

- The output of Sigmoid function is probability of association to a class e.g. if we evaluate and get $f(z) = 0.8$ for given feature set then we can say that probability of z falling under class 1 is 0.8. This probability is called weights.
- C : The inverse regularization parameter to tackle underfitting (high bias) and overfitting (low bias)



Logistic Regression (cont.)

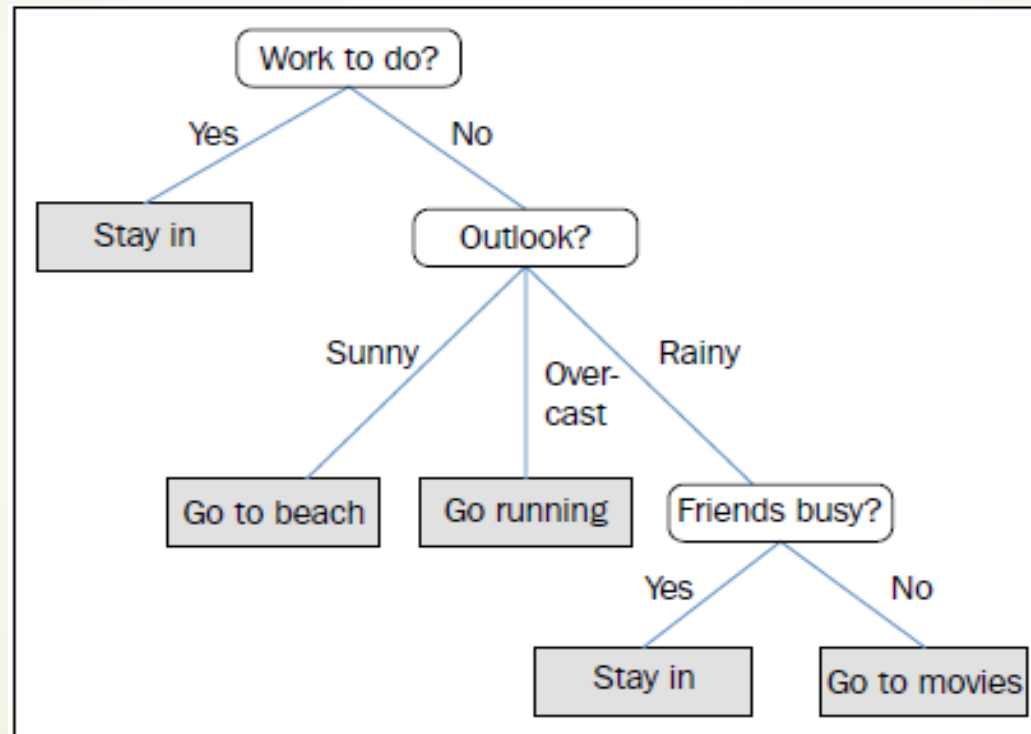
| C (Inverse Regularization Parameter) | Training Accuracy | Test Accuracy |
|--------------------------------------|-------------------|---------------|
| 1 | 0.780130 | 0.779221 |
| 0.01 | 0.773616 | 0.733766 |
| 100 | 0.778502 | 0.779221 |



Decision Tree

- We start at the tree root and split the data on the feature that results in the largest **information gain (IG)**. We can then repeat this splitting procedure until the leaves are pure (samples at each node belongs to the same class). This process can lead to overfitting. So, we need to set some limit for the depth of the tree.
- The information gain is simply the difference between the impurity of the parent node and the sum of the child node impurities—the lower the impurity of the child nodes, the larger the information gain.

Decision Tree (cont.)



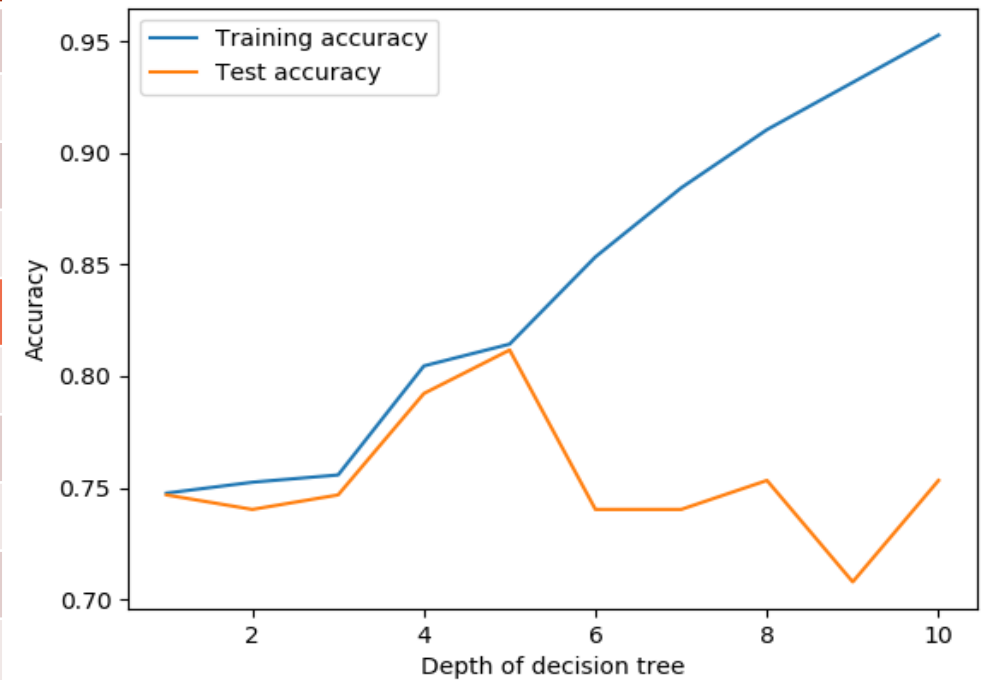


Decision Tree (cont.)

- ▶ Training Accuracy:1.000000 Test Accuracy:0.668831 (Tree grows until all leaves are pure)
- ▶ The training accuracy is 100% whereas test accuracy is only 67%. So we have overfitting here.
- ▶ Overfitting can be resolved by limiting the maximum depth of the decision tree.

Decision Tree (cont.)

| Max depth | Training accuracy | Test accuracy |
|-----------|-------------------|-----------------|
| 1 | 0.747557 | 0.746753 |
| 2 | 0.752443 | 0.740260 |
| 3 | 0.755700 | 0.746753 |
| 4 | 0.804560 | 0.792208 |
| 5 | 0.814332 | 0.811688 |
| 6 | 0.853420 | 0.740260 |
| 7 | 0.884365 | 0.740260 |
| 8 | 0.910423 | 0.753247 |
| 9 | 0.931596 | 0.707792 |
| 10 | 0.952769 | 0.753247 |



Decision Tree (cont.)

| Feature | Importance percentage |
|----------------------------|-----------------------|
| Glucose | 0.02393909 |
| Blood pressure | 0.49218275 |
| Skin thickness | 0.04224278 |
| Insulin | 0.04496367 |
| BMI | 0.17793698 |
| Diabetes Pedigree Function | 0.06376849 |
| Age | 0.15496624 |



Random Forest



- The idea behind ensemble learning is to combine **weak learners** to build a more robust model, a **strong learner**, that has a better generalization error and is less susceptible to overfitting.
- Steps:
 - Randomly choose n samples from training set. (bootstrap sample)
 - Grow a decision tree out of bootstrap with following conditions:
 - Randomly select d features without replacement
 - Split node using the best split according to the objective function. (maximize information gain)
 - Repeat the procedure for 1 to K times.
 - Aggregate the prediction by each tree to assign the class label by majority vote.



Random Forest (cont.)

- It is not as interpretable as single decision tree.
- Ensemble model is quite robust to noise.
- Large no of iterations provides more accuracy at expanse of computational cost.

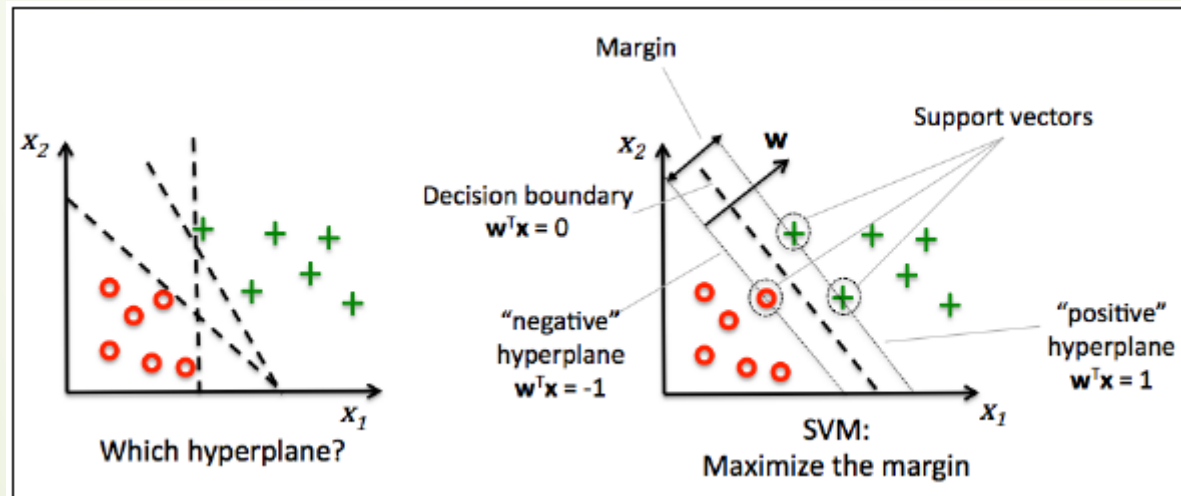


Random Forest (cont.)

- For 10 trees:
 - Training Accuracy: 0.980456 Test Accuracy: 0.766234
- For 100 trees with max depth of 5
 - **Training Accuracy: 0.819218 Test Accuracy: 0.831169**


Support vector machine

- The main objective in this technic is to maximize the distance between separating hyperplanes and the training samples (support vectors) that are closet to the hyperplane.



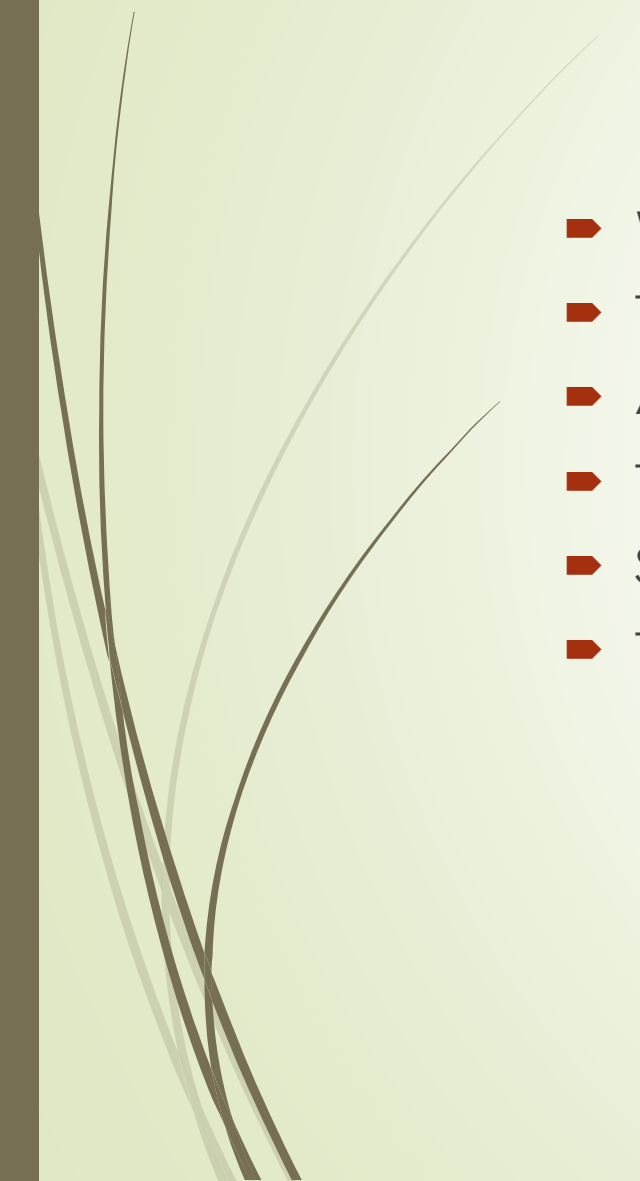


Support vector machine (cont.)

- 
- For data which is not linearly separable, a slack variable is used to relax linear constraints for nonlinearly separable data to allow convergence of the optimization in the presence of misclassifications under the appropriate cost penalization. This penalty cost can be varied to adjust bias-variance trade off.
 - If this cost is large there will be larger penalties for errors and if it is small, then it will be less strict for misclassification errors.



Support vector machine(cont.)

- Without pre-processing
 - Training Accuracy: 1.000000 Test Accuracy: 0.597403
 - After pre-processing and $C = 1$:
 - Training Accuracy: 0.842020 Test Accuracy: 0.727273
 - Setting penalty cost $C = 50$:
 - Training Accuracy: 0.928339 Test Accuracy: 0.740260
- 



Comparison of algorithms



| Algorithm | Optimal test accuracy |
|--------------------------------|-----------------------|
| K-Nearest Neighbours algorithm | 0.720779 |
| Logistic Regression | 0.733766 |
| Decision Tree | 0.811688 |
| Random Forest | 0.831169 |
| Support vector machine | 0.727273 |



Questions?



References



- Pima diabetes database, University of California, Irvine.
- Python Machine Learning by Sebastian Raschka
- Scikit-learn documentation