Student Names and IDs:

- Newton Kwan, nk150
- Ashka Stephen, aas74

Homework 7

Part 1: Random Guesses

Problem 1.1

What is the accuracy for a random-guess K-class classifier in the symmetric case? No need to prove your answer, if you are sure of it. Also give a numerical value *as an approximate percentage* with two decimal digits after the period when $K = 7$.

Answer

$$P(\hat{y} = y) = \frac{1}{K}$$

Numerical value = 14.29%

Problem 1.2

Write a formula for the accuracy of a random-guess K-class classifier in the general, zero-knowledge case. Also report a numerical value.

Answer

$$P(\hat{y} = y) = \frac{1}{K}$$

Numerical value: 14.29% (same as 1.1)

Problem 1.3

Write a formula for the accuracy of a random-guess K-class classifier in the fully general case. Also report a numerical value.

Answer

$$P(\hat{y} = y) = \sum_{i=1}^{k} q_i p_i$$

Numerical value: 8%

Problem 1.4

Write a formula for the accuracy of a random-guess K-class classifier in the general, perfect-knowledge case. Also report a numerical value.

Answer

$$P(\hat{y} = y) = \sum_{i=1}^{k} p_i^2$$

Numerical value: 24%

Problem 1.5

Is the classifier better off by having perfect knowledge rather than assuming equiprobable labels in the numerical examples given above? The formula for accuracy in the perfect-knowledge case is related to one of the measures of impurity we discussed in the context decision trees. Which measure, and how?

Answer

Yes perfect knowledge is better as we can see from the example problem (comparing 24% accuracing in perfect knowledge with approximately 14% accuracy assuming equal probabilities). This is related to the Gini index since the index calculates the probability of (1 - accuracy) in the perfect knowledge case.

$$i(S) = 1 - \sum_{y \in Y} p^2(y|S)$$

Part 2: Multiclass Classifiers

In [5]:
```
import numpy as np
from sklearn.datasets import fetch_covtype

# So we all look at the data the same way
seed = 3
np.random.seed(seed)

data = fetch_covtype(random_state=seed, shuffle=True)

# Random training/test data partition
from sklearn.model_selection import train_test_split
testFraction = 0.2
T, S = {}, {}

#x_train, x_test, y_train, y_test
T['x'], S['x'], T['y'], S['y'] = train_test_split(data.data, data.target
,
 test_size=testFraction, random_state=seed)
```

Problem 2.1

Use the function `LogisticRegression` from `sklearn.linear_model` to train a logistic-regression classifier on `T`. Use the following keyword parameters:

```
C=1e5, solver='lbfgs', multi_class='multinomial', random_state=seed
```

If you want to know what these mean (a good idea), read the manual.

Show your code and report the training accuracy and the test accuracy as a *percentage* (not a fraction) with two decimal digits after the period.

Parameter info:

- solver: handle multinomial loss only handle L2 penalty
- multiclass For ‘multinomial’ the loss minimised is the multinomial loss fit across the entire probability distribution, even when the data is binary. Stochastic Average Gradient descent solver multinomial = the softmax function is used to find the predicted probability of each class.
- random_state - The seed of the pseudo random number generator to use when shuffling the data

Answer

In [22]:
```
from sklearn.linear_model import LogisticRegression
def information(classifier, T, S, C, solver, multi_class, random_state):
    '''
    Describes the result of evaluating the classifier on the sets
    '''
    clf = classifier(C=C, solver=solver, multi_class=multi_class, random
_state=random_state).fit(T['x'], T['y'])
    train_risk = clf.score(T['x'], T['y'])
    test_risk = clf.score(S['x'], S['y'])
    print("The training accuracy is {:.2f}".format(train_risk*100), "%")
    print("The test accuracy is {:.2f}".format(test_risk*100), "%")
    return

answer = information(LogisticRegression, T, S, 1e5, 'lbfgs', 'multinomia
l', seed)
```

```
The training accuracy is 61.92 %
The test accuracy is 61.61 %
```

Problem 2.2

Comment on the results you obtained in the previous problem:

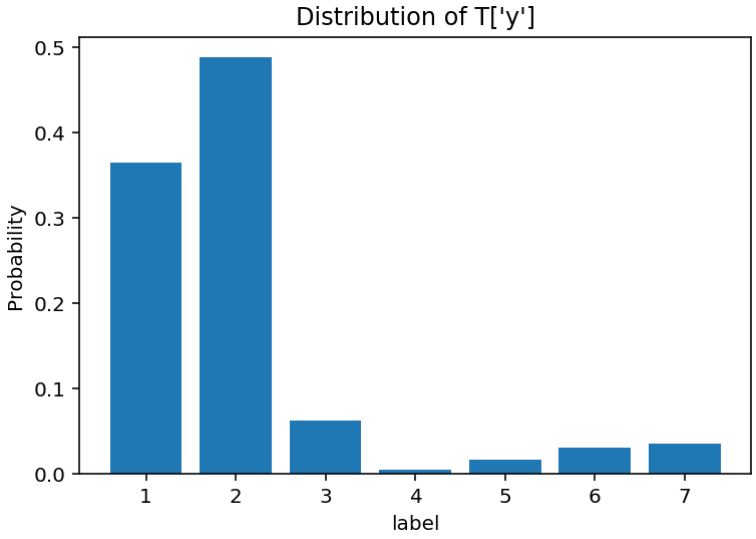
- Is performance good?
- What may be a plausible reason for this level of performance?
- Is there overfitting, and how can you tell?
- Is performance better than random guessing, with or without knowledge of the label distribution?

Assume that the label distribution estimated from `T` constitutes ''perfect knowledge.'' Include accuracies as percentages (with two decimal digits after the period) in your arguments. Don't forget to answer any of the questions, and answer them in bullet form, in the order they are asked.

Answer

In [23]:
```
# create a dictionary that stores how many of each label is in training
 set, T
d = {}
num_labels = len(T['y']) # size of training set
for i in T['y']:
    if i not in d:
        d[i] = 1
    else:
        d[i] += 1
print("Dictionary of T:", d) #
print()

# create the probability distribution of T
for key in d:
    d[key] /= num_labels
print("Distribution of T:", d)

import numpy as np
import matplotlib
import matplotlib.pyplot as plt
%precision %g
%config InlineBackend.figure_format = 'retina'
matplotlib.rcParams['savefig.dpi'] = 120
matplotlib.rcParams['figure.dpi'] = 120
%matplotlib inline

x_list = list(d.keys())
y_list = list(d.values())
plt.bar(x_list, y_list, align = 'center')
plt.title("Distribution of T['y']")
plt.xlabel("label")
plt.ylabel("Probability")

K = len(d.keys())
rand_no_labels = 1 / K
rand_with_labels = 0
for key in d:
    rand_with_labels += d[key]**2

print("Random guessing with no labels {:.2f}".format(rand_no_labels*100
), "%")
print("Random guessing with labels {:.2f}".format(rand_with_labels*100),
 "%")
```

```
Dictionary of T: {3: 28570, 2: 226673, 1: 169515, 5: 7596, 7: 16407, 6:
13875, 4: 2173}

Distribution of T: {3: 0.06146610758397535, 2: 0.4876691286098161, 1:
0.3646981878578083, 5: 0.016342196472099293, 7: 0.03529836986805333, 6:
0.029850971044020232, 4: 0.004675038564227457}
Random guessing with no labels 14.29 %
Random guessing with labels 37.70 %
```



- Is performance good? Test accuracy is fairly low, at ~61.62%, which we think (in somewhat of an arbitrary way) means that performance is not good.
- What may be a plausible reason for this level of performance? One plausible reason for this level of performance is that the distribution of the data may not be linearly separable. For example, in R^2, if the data is shaped like a circle, there is no way to separate the data well with a decision boundary that is a line. (In the example case, a decision boundary that is a circle may work better)
- Is there overfitting, and how can you tell? We see that the test and training accuracies are similar in value. If training accuracy was significantly higher than the test accuracy, we would be able to conclude that training overfits; however in this case there is no overfitting since the accuracies are almost equivalent.
- Is performance better than random guessing, with or without knowledge of the label distribution? The test accuracy of logistic regression is 61.62 %. The test accuracy of random guessing without labels is 14.29 %. The test accuracy of random guessing with labels is 37.70%. Performance using logistic regression on the test set (61.62%) is better than random guessing with knowledge of the label distribution (37.70%) and better than random guessing without knowledge of the label distribution (14.29%)

Problem 2.3

Now try a `DecisionTreeClassifier` from `sklearn.tree` on the same problem. Show your code and report training and test accuracy. Use default parameters.

Answer

In [24]:
```
from sklearn.tree import DecisionTreeClassifier
def information(classifier, T, S, random_state):
    '''
    Desribes the result of evaluating the classifier on the sets
    '''

    clf = classifier(random_state=random_state).fit(T['x'], T['y'])
    train_risk = clf.score(T['x'], T['y'])
    test_risk = clf.score(S['x'], S['y'])
    print("The training accuracy is {:.2f}".format(train_risk*100), "%")
    print("The test accuracy is {:.2f}".format(test_risk*100), "%")
    return


answer = information(DecisionTreeClassifier, T, S, seed)
```

```
The training accuracy is 100.00 %
The test accuracy is 93.86 %
```

Problem 2.4

Discuss the decision-tree results in the same way you did for the linear classifier. Don't forget to answer any of the questions, and answer them in bullet form, in the order they were asked.

Answer

- Is performance good? Performance is great! Our test accuracy is ~93%, which we deem as good performance.
- What may be a plausible reason for this level of performance? Decision trees usually fit data well as they separate data into categories until leaves are “pure”. Therefore, whether the data is linearly separable or not doesn’t matter, as any item is classified by the label of the pure leaf it fits into.
- Is there overfitting, and how can you tell? Yes, because the training score is 100%, meaning that the training data is perfectly fitted. However, this is not necessarily bad because, in our case, we still have a high test accuracy value.
- Is performance better than random guessing, with or without knowledge of the label distribution? The test accuracy of the Decision Tree is 93.86%. The test accuracy of random guessing without labels is 14.29%. The test accuracy of random guessing with labels is 37.70%. Therefore we see that performance using logistic regression on the test set (93.86%) is better than random guessing with knowledge of the label distribution (37.70%) and better than random guessing without knowledge of the label distribution (14.29%). Performance is better than random guessing with knowledge of label distribution because the label distribution is what allows us to create the decision tree and to therefore perform well on the test set.

Part 3: Cross-Validation

Problem 3.1

Use `GridSearchCV` from `sklearn.model_selection` to perform 10-fold cross-validation in order to determine good hyper-parameters for the decision tree you developed in the previous part, and with the same data. Find the best out of all possible combinations of the following parameters:

```
'criterion': ['gini', 'entropy']
'max_depth': range(30, 50, 5)
'min_samples_leaf': [1, 10, 100, 1000]
```

Report the values of the best parameters found and the training and test accuracy as before.

In [20]:
```
from sklearn.model_selection import GridSearchCV

K = 10
#create dict
myDict = {
    'criterion': ['gini', 'entropy'],
    'max_depth': range(30, 50, 5),
    'min_samples_leaf': [1, 10, 100, 1000]
}

#Exhaustive search over specified parameter values for an estimator.
clf_kfold = GridSearchCV(DecisionTreeClassifier(), myDict, cv=K)
clf_kfold.fit(T['x'], T['y'])
best = clf_kfold.best_params_
print(best)

n_clf_kfold = DecisionTreeClassifier(criterion = best['criterion'], max_
depth = best['max_depth'], min_samples_leaf = best['min_samples_leaf'],
random_state=seed)
n_clf_kfold.fit(T['x'], T['y'])

training_score = n_clf_kfold.score(T['x'], T['y'])
test_score = n_clf_kfold.score(S['x'], S['y'])
print("The training accuracy is {:.2f}".format(training_score*100), "%")
print("The test accuracy is {:.2f}".format(test_score*100), "%")
```

```
{'criterion': 'entropy', 'max_depth': 40, 'min_samples_leaf': 1}
The training accuracy is 100.00 %
The test accuracy is 94.45 %
```

Answer

Problem 3.2

Discuss your results:

- Is there significant improvement, or any improvement at all?
- Give a plausible explanation for the result

Answer

We see that without cross validation, our test accuracy is 93.86%, while with cross validation our testing accuracy increases to 94.45%. Thus although cross validation does improve the test accuracy, the improvement is not very significant. Additionally, it is important to consider that cross validation took about an hour to run, resulting in one percent increase in accuracy.

One possible explanation for this result is the max_depth is bounded with cross validation, while without cross validation the decision tree is constructed so that all the leaves are pure. Because of this, more testing inaccuracies could occur with cross validation's bound. Another possible explanation could be that our cross validation used entropy instead of gini for the criterion.